

Real-Time Detection of Covert Channels in Highly Virtualized Environments

Anyi Liu, Jim Chen, Li Yang

▶ **To cite this version:**

Anyi Liu, Jim Chen, Li Yang. Real-Time Detection of Covert Channels in Highly Virtualized Environments. 5th International Conference Critical Infrastructure Protection (ICCIP), Mar 2011, Hanover, NH, United States. pp.151-164, 10.1007/978-3-642-24864-1_11 . hal-01571783

HAL Id: hal-01571783
https://inria.hal.science/hal-01571783

Submitted on 3 Aug 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Chapter 11

REAL-TIME DETECTION OF COVERT CHANNELS IN HIGHLY VIRTUALIZED ENVIRONMENTS

Anyi Liu, Jim Chen and Li Yang

Abstract Despite extensive research, covert channels are a principal threat to information security. Covert channels employ specially-crafted content or timing characteristics to transmit internal information to external attackers. Most techniques for detecting covert channels model legitimate network traffic. However, such an approach may not be applicable in dynamic virtualized environments because traffic for modeling normal activities may not be available.

This paper describes Observer, a real-time covert channel detection system. The system runs a secure virtual machine that mimics the vulnerable virtual machine so that any differences between two virtual machines can be identified in real time. Unlike other detection systems, Observer does not require historic data to construct a model. Experimental tests demonstrate that Observer can detect covert channels with a high success rate and low latency and overhead.

Keywords: Virtualized environments, covert channels, detection

1. Introduction

The widespread deployment of firewalls and other perimeter defenses to protect enterprise information systems has raised the bar for malicious external attackers. However, these defensive mechanisms are ineffective against insiders, who can access sensitive data and send it to external entities using secret communication channels. According to the Computer Security Institute [13], the percentage of the insider attacks rose to 59% in 2007. Insider attacks have overtaken viruses and worms as the most common type of attack [3].

A covert channel is a communication channel that can be exploited by a process to transfer information in a manner that violates system security policies. A successful covert channel leaks information to external entities in a manner

that is often difficult to detect. Researchers have proposed a variety of approaches to detect and prevent covert channels. Most covert channel detection approaches [4, 6, 7, 11, 26] construct models based on clean traffic and detect covert channels by searching for deviations in actual traffic. Upon detecting a covert channel, a variety of countermeasures [10, 16, 17, 30] can be applied to manipulate traffic and prevent information leaks.

While the approaches are effective against most covert channels, they require a sufficient amount of clean traffic. However, in networked virtual environments, such as those encountered in cloud computing, creating suitable models from clean historic traffic is problematic, mainly because the traffic associated with most virtual machine services is highly dynamic in nature. For example, virtual machines may migrate arbitrarily across virtual networks, revert to the snapshot of a saved state, or may run multi-booting systems. In such cases, clean historic traffic is either unavailable or the available traffic does not reflect the characteristics of clean traffic.

To address these challenges, we have designed and implemented the Outbound Service Validator (Observer), a real-time covert channel detection system. Observer leverages a secure virtual machine to mimic the behavior of a vulnerable virtual machine. It redirects all inbound traffic destined to the vulnerable virtual machine to the secure virtual machine, and differentiates between the outbound traffic of the two virtual machines to detect covert channels. Unlike existing approaches, Observer operates in real time and does not require historic traffic for modeling normal behavior. It can be dynamically incorporated in a cloud infrastructure when a vulnerable virtual machine has been identified. Moreover, the implementation is transparent to external attackers, which minimizes the risk that the detection system itself is the target of subversion. Experimental tests demonstrate that Observer can detect covert channels with a high success rate and low overhead. In particular, it induces an average 0.05 ms latency in the inter-packet delay and an average CPU usage increase of about 35% in a virtual network with 100 Mbps throughput.

2. Related Work

A number of differential analysis approaches have been developed for intrusion detection. Our approach is closely related to that of Netspy [31], which compares outgoing packets from a clean system with those from an infected system. However, our approach represents an advancement over Netspy's approach. First, Netspy detects spyware that leaks private information as plaintext in HTTP responses; in contrast, our approach detects stealthy attacks, such as covert channels, that involve information leaks using encrypted traffic. Second, in order to generate signatures, Netspy assumes that spyware generates additional network traffic from the infected system; this assumption fails when information is transmitted through a passive covert channel that does not generate extra traffic. Third, Netspy correlates inbound packets with the corresponding outbound packets that are triggered; this approach fails to de-

tect sophisticated covert channels that postpone outbound packets to ensure that inbound and outbound packets cannot be correlated.

Privacy Oracle [15] employs an approach similar to ours to discover information leaks. It uses perturbed user input to identify the fields of network traces by aligning pairs of network traces. Siren [5] uses crafted user input along with a description of legitimate user activities to thwart mimicry attacks. However, both Privacy Oracle and Siren are unable to differentiate anomalous output in order to detect covert channels.

The approach of Mesnier, *et al.* [20] predicts the performance and resource usage of a device using a mirror device. However, this approach is designed to predict the workload characteristics of I/O devices. Our approach, on the other hand, deals with the more difficult problem of detecting covert channels.

A number of methodologies have been proposed for creating covert channels [6, 12, 18, 27] and for detecting covert channels [1, 11, 19, 23]. The accuracy of these techniques depends on the availability of a good model and a substantial quantity of clean historic traffic. Our approach is superior in that it works online and neither requires modeling nor clean historic traffic. Moreover, it can be deployed dynamically or migrated across a networked virtual infrastructure, which renders it an attractive solution for highly virtualized environments.

Several researchers have applied covert channel design schemes to trace suspicious traffic. For example, Wang and Reeves [34] employ well-designed interpacket delays to trace suspicious VoIP traffic [32, 33]. In contrast, our work only focuses on detecting covert channels, although covert channels design schemes nicely complement our detection methodology.

Research efforts related to cross virtual machine covert/side channels [22, 25, 35] and their countermeasures [14] are relevant to our work. However, they deal with covert channels that leak information between virtual machines that share the same virtual machine monitor or hardware. Our work goes beyond inter-virtual-machine covert/side channels – it focuses on detecting aggressive covert channels between insiders and external entities.

3. Threat Model

Covert channels can be roughly categorized into two types: covert storage channels that manipulate the contents of storage locations (e.g., disk, memory, packet headers, etc.) and covert timing channels that manipulate the timing or ordering of events (e.g., disk accesses, memory accesses, inter-packet delays, etc.). This paper focuses on the detection of covert storage channels.

Despite the complex nature of networked virtual environments, we desire to handle covert channel threats in as general a manner as possible. First, we assume that a vulnerable virtual machine can be compromised by many exploits. These include exploits that target vulnerable services, zero-day attacks and internal subversion exploits; however, we exclude attacks that change virtual machine behavior via a virtual machine monitor or hypervisor. Second, we assume that, after a virtual machine has been compromised, its user-space applications and kernel-space device drivers can be fully controlled by the at-

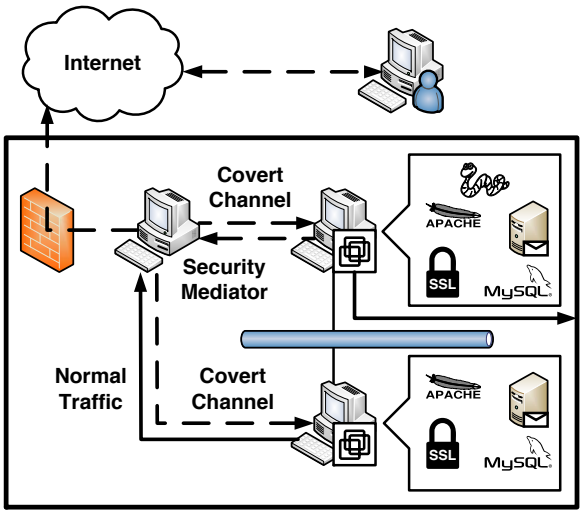


Figure 1. Covert channel threat.

tacker. Since covert channels leak internal information to external attackers, they can be detected by examining outbound network traffic. Figure 1 illustrates the covert channel threat.

The security foundation is based on two assumptions. First, the virtual machine monitor, which is under the control of the current virtual computing environment, is trusted and cannot be breached. Second, there exist a number of secure virtual machines that are also under the control of the current virtual computing environment. The secure virtual machines may be created along with the vulnerable virtual machine by cloning them from a clean state, or they may be created from a virtual machine prototype such as Amazon's Elastic Compute Cloud (EC2) [2] or Microsoft Azure Services [21]. The secure virtual machines are protected by the virtual computing environment and external attackers cannot compromise them. Note that, although this approach requires at least one secure virtual machine to synchronize with a vulnerable virtual machine, the number of secure virtual machines is bounded by the number of vulnerable servers being monitored. Therefore, the Observer system can be applied to monitor as many vulnerable servers as desired when computational and storage resources become available. We do not require the virtual machine monitor of the virtual environment to know the software (i.e., operating system and applications) installed on the virtual machines, although this information is useful to determine the integrity of the virtual machines.

4. System Architecture

Figure 2 presents the architecture of the Observer system. The system has two main components: the security mediator and the virtual machine repos-

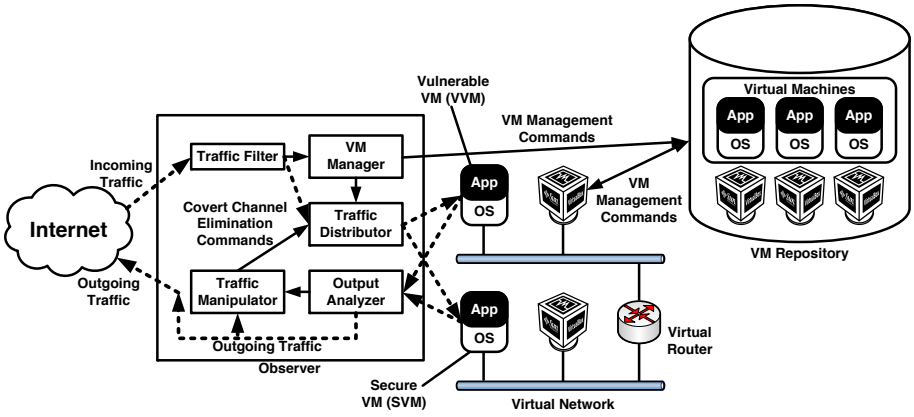


Figure 2. Observer system architecture.

itory. The security mediator comprises the: (i) traffic filter, which monitors inbound packets from the Internet and filters traffic of interest; (ii) traffic distributor, which examines the networking protocol information in the intercepted packets, replicates the packets and sends them to the vulnerable virtual machine and secure virtual machine; and (iii) output analyzer, which singles out outbound traffic from the two virtual machines and detects anomalous patterns. The virtual machine repository maintains a set of virtual machines, and activates, deactivates, clones and updates the virtual machine snapshots.

4.1 Traffic Filter

To protect services, the traffic filter maintains several rules that determine whether or not packets are to be intercepted. If an incoming packet satisfies a rule, it is subjected to further processing. After a new susceptible service is launched, new rules corresponding to the service are added to the rule list.

```
<?xml version="1.0" encoding="utf-8"?>
<root>
    <rule>
        <Action>INTERCEPT</Action>
        <Direction>OUT</Direction>
        <Protocol>TCP</Protocol>
        <From_IP>123.123.123.123</From_IP>
        <From_Port>ANY</From_Port>
        <To_IP>129.174.2.123</To_IP>
        <To_Port>80</To_Port>
        <Flags>NA</Flags>
    </rule>
    ...
    ...
    <rule>
        <Action>DROP</Action>
        <Direction>IN</Direction>
        <Protocol>TCP</Protocol>
        <From_IP>ANY</From_IP>
        <From_Port>ANY</From_Port>
        <To_IP>129.174.2.123</To_IP>
        <To_Port>80</To_Port>
        <Flags>TCPFLAG=rst</Flags>
    </rule>
</root>
```

Figure 3. Configuration file of traffic filter.

Figure 3 lists some rules, which are similar to general purpose firewall rules. The rules specify the packet header fields, such as *Direction*, *Protocol*, *Src_IP*, *Src_Port*, *Dst_IP*, *Dst_Port* and *Packet Header Flags*. The first rule specifies

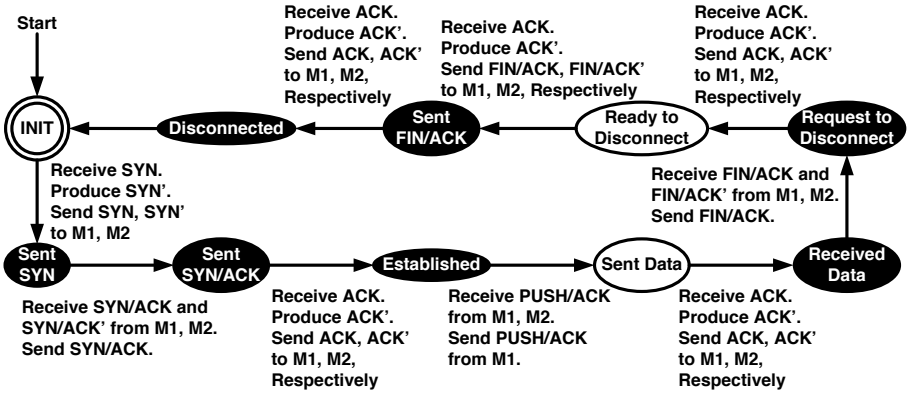


Figure 4. State machine for the traffic distributor.

that TCP packets from an external host (with address `123.123.123.123` and any port number) to an internal HTTP service (with address `129.174.2.123` and port number 80) are to be intercepted. The second rule drops all TCP packets in a reply to an external host, whose packet header field contains a `rst` flag. Rules can be added and deleted at runtime to ensure that the system cannot be penetrated during rule updates. Note also that the rules are based on *a priori* knowledge and reported vulnerabilities.

The time taken by the traffic filter is $O(N)$, where N is the total number of packets (in terms of the number of bytes) that satisfies the rule set provided that the focus is restricted to services that are easily compromised. Since the packet filter does not buffer processed packets, the storage requirement is bounded by the maximum size of a network packet.

4.2 Traffic Distributor

The primary task of the traffic distributor is to forward packets destined to the vulnerable virtual machine to the secure virtual machine. Two steps are involved. First, when the traffic distributor receives a packet e from the traffic filter, it constructs a new packet e'. The new packet e' keeps some of the fields from e (e.g., Src_IP and Src_Port), while other fields (e.g., Dst_IP, Dst_Port, sequence numbers and checksum) are modified (Dst_IP and Dst_Port correspond to the IP address and port number of the secure virtual machine, respectively). The two packets e and e' are then dispatched simultaneously. Second, when the traffic distributor receives reply packets from both the virtual machines, it only sends the packet reply corresponding to e; thus, an external attacker has no knowledge of the secure virtual machine.

Figure 4 presents a simplified state machine corresponding to the traffic distributor. $M1$ and $M2$ represent the vulnerable virtual machine and secure virtual machine, respectively. For example, when the traffic distributor receives a SYN packet, it constructs a new SYN packet, namely SYN', and sends

SYN and SYN' to $M1$ and $M2$, respectively. Similarly, when it receives the ACK and ACK' packets, it only sends the ACK packet. To synchronize the outputs of $M1$ and $M2$, the traffic distributor must maintain all the previous communication states in a queue in the event of packet loss or fragmentation.

The overhead involved in constructing a new packet e' is essentially constant. The storage requirement is dictated by the total number of packets forwarded from the traffic filter. Although Observer collects live traffic, which increases without bound, the storage requirement is still bounded by $2n$, where n is the size of the queue.

5. Implementation

Observer is implemented on a VMware ESX Server 4.1 [29]. The traffic filter and traffic distributor are implemented as part of a transparent bridge, which uses a customized `ipfw` application to intercept packets and `divert socket` to manipulate packets. The security mediator comprises around 1,000 lines of C code.

To minimize the latency after packets leave the security mediator, the vulnerable virtual machine and secure virtual machine are cloned from the same virtual machine image with the same state, and the two virtual machines are configured one hop away from Observer. The vulnerable virtual machine and secure virtual machine must be closely synchronized; the sequence number field of TCP packets is used to synchronize outbound traffic from the two virtual machines. To ensure that Observer maintains accurate time information, each virtual machine is configured to have affinity to one CPU at a time.

The output analyzer uses Ethereal to collect traffic and separate the timing information. The `ntop` application is used to generate network traffic statistics at runtime. The output analyzer module is written in C, Perl, Dataplot and MATLAB.

6. Evaluation

This section analyzes the ability of Observer to detect covert channels. It also examines the performance overhead involved in covert channel detection.

6.1 Covert Channel Construction and Detection

A scheme similar to that used by Ramsbrock, *et al.* [24] was used to construct covert channels. Specifically, to encode an i-bit sequence $S = s_0, \ldots, s_{i-1}$, we used $2i$ randomly chosen packet pairs $\langle P_{r_i}, P_{e_i} \rangle$ $(i = 0, \ldots, L)$ where $r_i \leq e_i$, and P_{r_i} and P_{e_i} correspond to reference packets and encoding packets, respectively. A covert bit s_k $(0 \leq k \leq i-1)$ is encoded into the packet pair $\langle P_{r_i}, P_{e_i} \rangle$ using the equation:

$$e(L_r, L_e, L, s_k) = l_e + [(0.5 + s_k)L - (l_e - l_r)] mod 2L$$

where L_r and L_e are the values of the encoded field in P_{r_i} and P_{e_i}, respectively.

Table 1. Detection window size for various covert channels.

	BTWC	PFC
a = 1	160	60
a = 5	181	460
a = 10	7,150	390
a = 20	24,060	730

The original covert channel design scheme was extended to test the effectiveness of Observer to detect slow covert channels. Specifically, instead of using $2i$ packets, we used $2ai$ ($a \geq 1$) packets to encode S, where a is a constant or pseudorandom number. P_{r_i} and P_{e_i} were chosen from the $2a$ packets. The term a serves as an "amplifier," where a larger value indicates a slower covert channel.

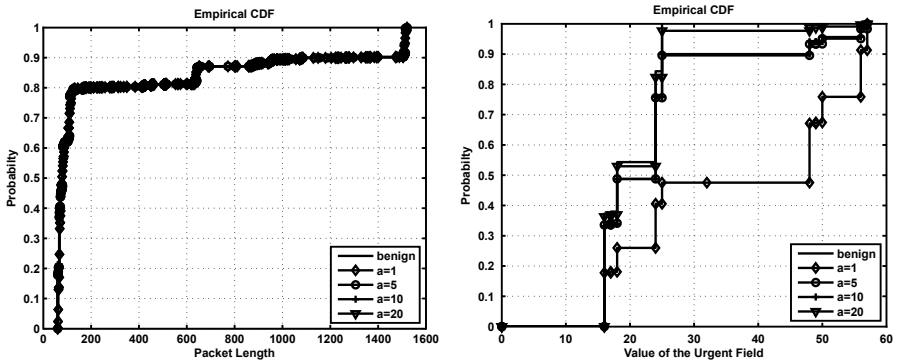
Covert channels were detected using shape tests corresponding to first-order statistics such as means and variances [23]. The shapes of the traffic patterns were tested using a Chi-Square test [8] and a two-sample Kolmogorov-Smirnov test (K-S test) [9]. The Chi-Square test was used to verify whether or not two discrete sample data come from the same discrete distribution; the K-S test was used to verify whether or not two continuous sample data come from the same continuous distribution. The two tests were chosen because they are distribution free, i.e., they do not depend on the actual cumulative distribution function (CDF) being tested.

6.2 Effectiveness

The evaluation focused on the detection of two types of covert channels: IP/TCP packet field channels (PFCs) and botnet traceback watermark channels (BTWCs). A PFC operates by modifying the *urgent* field of TCP packets to transmit information: a `1` bit is transmitted by increasing the *urgent* field value by an integer modulo w, while a `0` bit is transmitted by increasing the value by an integer modulo $\lfloor \frac{w}{2} \rfloor$. A BTWC operates by modifying the length of the encoding packet P_{e_i} by padding characters to achieve a specific length that is different from its corresponding reference packet P_{r_i}. The padded characters could be visible or invisible (e.g., whitespace), and can be inserted in random locations in the payload.

The latency of Observer was measured using the detection window size, which is the minimum number of packets needed to detect a covert channel since it commenced transmission. A larger window size indicates greater latency and less sensitivity to covert channels in real time. Table 1 shows the detection window size required by Observer to obtain a 100% true positive rate (note that a larger value of a indicates a slower covert channel). The results demonstrate that a slower covert channel requires a larger detection window.

Table 1 shows that that a window size of 160 is required to detect the most aggressive BTWC, which sends one bit per packet. However, a slower BTWC,



(a) Packet length distributions.

(b) *Urgent* field distributions.

Figure 5. Cumulative distributions for BTWCs and PFCs for various values of a.

which transmits one bit every 20 packets, requires a window size of 24,060! The detection window sizes, which are much more sensitive for PFCs, vary from 60 to 730.

The results can be explained by comparing the cumulative distribution functions (CDFs) corresponding to BTWCs and PFCs for different values of a. Figure 5 shows that the distributions of BTWCs are quite similar for different values of a, but the same is not true for PFCs. Therefore, BTWCs are more difficult to detect than PFCs.

6.3 Detection Rate

Live traffic was collected from the vulnerable virtual machine and secure virtual machine in order to determine the false positive rates for Observer. A BTWC and PFC were created with $a = 1$; the false positive rates for both channels were zero. Tables 2 and 3 present the results of the statistical tests for the two channels. As expected, the slower covert channels (with larger values of a) have statistics that are closer to those of legitimate traffic.

A theoretic analysis of false positives was conducted by setting the targeted false positive rate to 1%. To achieve this false positive rate, we used a "cut-off point," which was set at the 99^{th} percentile of the legitimate samples to determine if samples are benign or malicious.

Figure 6 shows the true positive rates for PFC and BTWC detection for various values of a. The results show that the effectiveness of detection depends on the number of observed packets. For example, in the case of a PFC with $a = 10$, the true positive rate fluctuates when Observer collects 5,000 packets. A similar situation also occurs for a BTWC. We are currently investigating various approaches to improve the effectiveness of detection.

Table 2. PFC test scores.

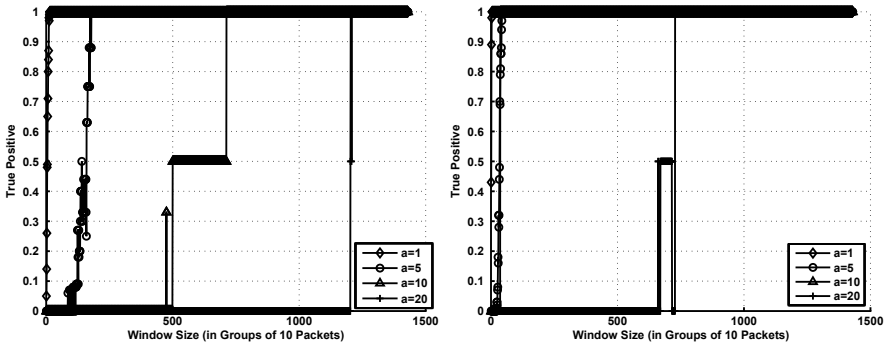
	Mean	SD	Chi-Square Test	Chi-Square (CDF value)	Chi-Square (1% cutoff)
Legitimate HTTP	20.143	3.925	0	0	≥ 15.087
PFC (a = 1)	36.926	16.485	6789.002	1	≥ 11.345
PFC (a = 5)	23.480	10.576	82.136	1	≥ 15.086
PFC (a = 10)	23.343	10.316	75.657	1	≥ 13.277
PFC (a = 20)	20.878	6.137	3.902	0.581	≥ 13.277
PFC (a = 50)	20.478	5.088	0.796	0.061	≥ 13.277

Table 3. BTWC test scores.

	Mean	SD	K-S Test	K-S (p value)	K-S (1% cutoff)
Legitimate HTTP	283.601	453.532	0	1	≥ 0.1923
BTWC (a = 1)	284.785	452.005	0.248	0	≥ 0.1923
BTWC (a = 5)	283.824	453.215	0.049	2.94e-15	≥ 0.1923
BTWC (a = 10)	283.721	453.357	0.025	2.53e-04	≥ 0.1923
BTWC (a = 20)	283.660	453.441	0.012	0.212	≥ 0.1929
BTWC (a = 50)	283.622	453.503	0.004	0.951	≥ 0.1929

6.4 Performance

Two experiments were conducted to evaluate the performance of Observer. The first experiment evaluated the throughput of Observer under the best and worst case scenarios. In the best case scenario, no packets were intercepted by the traffic filter. In the worst case scenario, the majority of the inbound packets sent to the vulnerable virtual machine were intercepted by the traffic filter because they were assumed to be sent via a covert channel. Packets in both scenarios were collected when an attacker visited the web server of the vulnerable virtual machine for 160 minutes. Table 4 presents the detailed statistics for the two scenarios. Note that the average time to process a packet is almost 56.2 ms even in the worst case.



(a) PFC true positive rates. (b) BTWC true positive rates.

Figure 6. True positive rates of channel detection for various values of a.

Table 4. Comparison of throughput.

	Best Scenario	**Worst Scenario**	**Ratio**
Packets	1,802,176,469	170,712,403	0.094
Bytes	1,214,518,973,379	128,003,547,066	0.105
Packets/s	187,727	17,783	0.098
Bytes/s	126,512,393	13,333,703	0.110
Duration	160 min	160 min	1.000

The second experiment measured the average latency introduced by Observer to the inter-packet delay. In the experiment, 1,000,000 packets were collected with and without Observer installed. The average latency added to the inter-packet delay is 5 ms, which is almost unnoticeable compared with the reported average inter-packet delay of 42.67 ms for Northern America [28]. This latency is the result of the queuing delay that Observer imposes on inbound packets when directing them to both virtual machines at the same time.

7. Conclusions

Observer detects covert channels in a networked virtual environment by running a secure virtual machine that mimics a vulnerable virtual machine so that differences between the two virtual machines can be identified. Unlike most covert channel detection systems, Observer does not rely on historic data to create models of normal behavior. Experimental tests demonstrate that Observer can detect covert channels with a high success rate and low latency and overhead.

The accuracy of covert channel detection relies on the fact that the virtual computing environment provides at least one secure virtual machine for every

vulnerable virtual machine. This limits the scalability of our approach. To address this limitation, our future research will investigate the dynamic allocation and management of secure virtual machines.

Another limitation is that it is necessary to maintain a secure version of a vulnerable virtual machine over its entire lifecycle. Even during the detection phase, it is difficult to ensure that inbound traffic does not contain an exploit that could compromise the secure virtual machine at runtime. Potential solutions that address this limitation will be examined in our future research.

References

[1] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, Trojan detection using IC fingerprinting, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 296–310, 2007.

[2] Amazon, Amazon Elastic Compute Cloud (Amazon EC2), Seattle, Washington (aws.amazon.com/ec2).

[3] M. Ben Salem, S. Hershkop and S. Stolfo, A survey of insider attack detection research, in *Insider Attack and Cyber Security: Beyond the Hacker*, S. Stolfo, S. Bellovin, S. Hershkop, A. Keromytis, S. Sinclair and S. Smith (Eds.), Springer, New York, pp. 69–90, 2008.

[4] V. Berk, A. Giani and G. Cybenko, Covert channel detection using process query systems, *Proceedings of the Second Annual Workshop on Flow Analysis*, 2005.

[5] K. Borders, X. Zhao and A. Prakash, Siren: Catching evasive malware, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 78–85, 2006.

[6] S. Cabuk, Network Covert Channels: Design, Analysis, Detection and Elimination, Ph.D. Thesis, Department of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, 2006.

[7] S. Cabuk, C. Brodley and C. Shields, IP covert timing channels: Design and detection, *Proceedings of the Eleventh ACM Conference on Computer and Communications Security*, pp. 178–187, 2004.

[8] G. Corder and D. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*, John Wiley, Hoboken, New Jersey, 2009.

[9] R. Duda, P. Hart and D. Stork, *Pattern Classification*, John Wiley, New York, 2001.

[10] G. Fisk, M. Fisk, C. Papadopoulos and J. Neil, Eliminating steganography in Internet traffic with active wardens, *Proceedings of the Fifth International Workshop on Information Hiding*, pp. 18–35, 2002.

[11] S. Gianvecchio and H. Wang, Detecting covert timing channels: An entropy-based approach, *Proceedings of the Fourteenth ACM Conference on Computer and Communications Security*, pp. 307–316, 2007.

[12] S. Gianvecchio, H. Wang, D. Wijesekera and S. Jajodia, Model-based covert timing channels: Automated modeling and evasion, *Proceedings of the Eleventh International Symposium on Recent Advances in Intrusion Detection*, pp. 211–230, 2008.

[13] L. Gordon, M. Loeb, W. Lucyshyn and R. Richardson, 2006 CSI/FBI Computer Crime and Security Survey, Computer Security Institute, San Francisco, California, 2006.

[14] T. Jaeger, R. Sailer and Y. Sreenivasan, Managing the risk of covert information flows in virtual machine systems, *Proceedings of the Twelfth ACM Symposium on Access Control Models and Technologies*, pp. 81–90, 2007.

[15] J. Jung, A. Sheth, B. Greenstein, D. Wetherall, G. Maganis and T. Kohno, Privacy Oracle: A system for finding application leaks with black box differential testing, *Proceedings of the Fifteenth ACM Conference on Computer and Communications Security*, pp. 279–288, 2008.

[16] M. Kang, I. Moskowitz and S. Chincheck, The pump: A decade of covert fun, *Proceedings of the Twenty-First Annual Computer Security Applications Conference*, pp. 352–360, 2005.

[17] M. Kang, I. Moskowitz and D. Lee, A network version of the pump, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 144–154, 1995.

[18] B. Kopf and D. Basin, An information-theoretic model for adaptive side-channel attacks, *Proceedings of the Fourteenth ACM Conference on Computer and Communications Security*, pp. 286–296, 2007.

[19] Y. Liu, C. Corbett, K. Chiang, R. Archibald, B. Mukherjee and D. Ghosal, SIDD: A framework for detecting sensitive data exfiltration by an insider attack, *Proceedings of the Forty-Second Hawaii International Conference on System Sciences*, 2009.

[20] M. Mesnier, M. Wachs, B. Salmon and G. Ganger, Relative fitness models for storage, *ACM SIGMETRICS Performance Evaluation Review*, vol. 33(4), pp. 23–28, 2006.

[21] Microsoft, Microsoft Azure Services Platform, Redmond, Washington (www.microsoft.com/azure/default.mspx).

[22] K. Okamura and Y. Oyama, Load-based covert channels between Xen virtual machines, *Proceedings of the Twenty-Fifth Symposium on Applied Computing*, pp. 173–180, 2010.

[23] P. Peng, P. Ning and D. Reeves, On the secrecy of timing-based active watermarking trace-back techniques, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 334–349, 2006.

[24] D. Ramsbrock, X. Wang and X. Jiang, A first step towards live botmaster traceback, *Proceedings of the Eleventh International Symposium on Recent Advances in Intrusion Detection*, pp. 59–77, 2008.

[25] T. Ristenpart, E. Tromer, H. Shacham and S. Savage, Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds, *Proceedings of the Sixteenth ACM Conference on Computer and Communications Security*, pp. 199–212, 2009.

[26] G. Shah, A. Molina and M. Blaze, Keyboards and covert channels, *Proceedings of the Fifteenth USENIX Security Symposium*, pp. 59–75, 2006.

[27] R. Smith and G. Scott Knight, Predictable design of network-based covert communication systems, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 311–321, 2008.

[28] Verizon, IP latency statistics, New York (www.verizonbusiness.com/about/network/latency), 2010

[29] VMware, VMware ESXi and ESX Info Center, Palo Alto, California (www.vmware.com/products/vsphere/esxi-and-esx/index.html).

[30] M. Vutukuru, H. Balakrishnan and V. Paxson, Efficient and robust TCP stream normalization, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 96–110, 2008.

[31] H. Wang, S. Jha and V. Ganapathy, NetSpy: Automatic generation of spyware signatures for NIDS, *Proceedings of the Twenty-Second Annual Computer Security Applications Conference*, pp. 99–108, 2006.

[32] X. Wang, S. Chen and S. Jajodia, Tracking anonymous peer-to-peer VoIP calls on the Internet, *Proceedings of the Twelfth ACM Conference on Computer and Communications Security*, pp. 81–91, 2005.

[33] X. Wang, S. Chen and S. Jajodia, Network flow watermarking attack on low-latency anonymous communication systems, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 116–130, 2007.

[34] X. Wang and D. Reeves, Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays, *Proceedings of the Tenth ACM Conference on Computer and Communications Security*, pp. 20–29, 2003.

[35] Z. Wang and R. Lee, Covert and side channels due to processor architecture, *Proceedings of the Twenty-Second Annual Computer Security Applications Conference*, pp. 473–482, 2006.