Design, Implementation and Evaluation of Covert Channel Attacks

Hamed Okhravi†
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2918
Email: hamed.okhravi@ll.mit.edu

Stanley Bak
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2302
Email: sbak2@illinois.edu

Samuel T. King
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2302
Email: kingst@illinois.edu

***Abstract*—Covert channel attacks pose a threat to the security of critical infrastructure and key resources (CIKR). To design defenses and countermeasures against this threat, we must understand all classes of covert channel attacks along with their properties. Network-based covert channels have been studied in great detail in previous work, although several other classes of covert channels (hardware-based and operating system-based) are largely unexplored. One of our contributions is investigating these classes by designing, implementing, and experimentally evaluating several specific covert channel attacks. We implement and evaluate hardware-based and operating system-based attacks and show significant differences in their properties and mechanisms. We also present channel capacity differences among the various attacks, which span three orders of magnitude. Furthermore, we present the concept of *hybrid* covert channel attacks which use two or more communication categories to transport data. *Hybrid* covert channels can be qualitatively harder to detect and counter than traditional covert channels. Finally, we summarize the lessons learned through covert channel attack design and implementation, which have important implications for critical asset protection and risk analysis. The study also facilitates the development of countermeasures to protect CIKR systems against covert channel attacks.**

I. Introduction

Whenever system builders design secure systems, they build these systems starting with a set of assumptions – one common way to break secure systems is by violating these assumptions. This principle is illustrated clearly by covert channel (CC) attacks where two entities can communicate by manipulating shared resources in unintended ways, endangering critical assets. Attackers can use such a mechanism to leak sensitive information, thus violating "provably correct" information flow policies. As a result, understanding CCs is paramount to improving the cyber security of CIKR systems.

Network-based CCs are a well-studied class of CCs. They can exploit unused or nonsensitive fields of network packets (e.g., time to live or IP options), protocol specification and properties (e.g., route updates or splitting algorithms), or timing of packets (e.g., inter-arrival time or jitter) to transmit bits of data covertly. Previous research has focused on new attack design, analysis of existing covert channels, and covert channel defense. However, previous efforts present mostly information theoretic analysis of covert channels to provide channel capacity estimation with few implementations and experimental results. Furthermore, non-network classes of CCs have received less attention recently. Clear classification and experimental results give system builders a good starting point for defense against CCs.

In this paper, we examine previously under-explored classes of CCs through design and implementation. Upon implementation, we evaluate empirically each channel's bit rate, noise and capacity. These measurements provide guidance in determining which parts of the system are likely to contain high-bandwidth CCs.

In addition, we introduce the idea of a *hybrid covert channel* that combines two or more classes of communication into a single CC. In CCs, there are typically three communication steps. First, a *send* mechanism emits bits of data by manipulating a shared resource. Then, a *receive* mechanism in a different process infers the bits by monitoring a shared resource. Finally, an optional *feedback* mechanism back to the sender provides direct synchronization and reduces noise. In previous work, the send, receive, and feedback mechanisms, for a given attack, use the same communication mechanism. For instance, a sender and receiver using the network may both read or write in unused IP bit fields. In contrast, a hybrid covert channel uses different communication mechanisms within a single attack. For example, a hybrid covert channel may have a send mechanism which uses the network and a receive mechanism that uses the operating system (we implement a sample attack). We argue that a well-designed hybrid CC can be more difficult to

†The author is currently an MIT Lincoln Laboratory employee.

reason about and defend against than a traditional CC.

Experimental results demonstrate that different CCs exhibit radically different channel capacities. We demonstrate that the maximum capacity of our implemented attacks can vary by three orders of magnitude. Moreover, we show that hybrid CC attacks are possible with less privileges than that of classic CCs. We implement a hybrid CC where, although access to the sending resource is blocked, the receiver still obtains data from the sender.

The contributions of this paper include: 1) providing the designs and implementations of several previously unexplored classes of CCs, 2) introducing the concept of a *hybrid* CC that combines different classes of communication within a single CC, and 3) demonstrating the practical differences among various classes of CCs and summarizing the most important lessons we learned during their design and implementation.

The rest of the paper is organized as follows. Section II describes the threat model used in our attacks. Section III specifies the classification scheme in detail along with the designs of new attacks in the unexplored classes of covert channels. Experimental evaluation of the CCs is detailed in Section IV. Section V explores the most important lessons learned. Related work is reviewed in Section VI and the paper is concluded in Section VII.

II. Threat Model

The general threat model in this paper assumes that there are two entities (e.g., processes) who want to share information illegitimately. The security policy prohibits these two entities from communicating directly. The security policy can be a simple multi-level security (MLS) policy or any general, arbitrary policy. However, for simplicity, we associate the sender with a "high" security clearance and the receiver with a "low" security clearance. The assumption is that the sender has access to some critical assets that the receiver does not and the sender intends to pass this information to the receiver covertly.

It is further assumed that the sender has "write" or "manipulation" access to the resource that is being used as the communication medium and the receiver has "read" or "monitoring" access. For example, for a filesystem-based CC, the sender must have permission to create a file and the receiver must have a way to monitor the existence of such a file (e.g., by directory listing).

Although this is the general threat model used in the paper, some of the attacks we design require weaker privileges. In those cases we state the less restrictive threat model after we explain the attack. In this paper, we do not deal with information leakages that happen between two entities inside a monolithic piece of code. Hence, the entities used here are always two different processes or threads.

III. Design and Implementation

In this section, we classify CCs based on the transmission mechanism and discuss the design and implementation of our five traditional CC attacks. We also introduce hybrid CCs and describe a sample implementation.

A. Covert Channel Classification

Security researchers classify CCs using a variety of schemes [2], [7], [11], [18]. Classically CCs have been divided into storage or timing channels [10]. Storage channels involve data that is presented or written to a storage location to transmit information. Timing channels send information in a way that involves manipulating the timing properties of a component of the system, such as by affecting the performance experienced by another process.

Modern computers can transmit timing and storage information using a variety of mechanisms. Most studies focus on network channels that can send or receive information over the network. Additionally, hardware devices, such as PCI peripherals, are shared among processes within a computer and can be used to transmit covert data. Moreover, operating systems maintain large amounts of data about the system and contain many data structures that can potentially be manipulated (directly or indirectly).

More recently, Wang and Lee divide CCs into value-based and transition-based channels [17]. The key distinction between these methods is that value-based channels transmit information based on the actual value present somewhere in the system, whereas transition-based channels use the change of a value to transmit information. The transition-based channels appear to be the more difficult to prevent, since all that is required is the ability to change a value to anything, not necessarily a specific value.

The classification scheme we use in this paper is motivated both by previous efforts to classify CCs and by CCs presented in literature. Our classification criteria are storage/timing, network/OS/hardware, and value/transition based. All three criteria are orthogonal to each other such that a particular attack my fall into any of the categories in any of the dimensions. Table I shows our classification scheme including citations for specific attacks and references to attacks we implement.

B. Hardware Storage Value/Transition-Based Attack

The first category of attack we study is a hardware, storage, value-based attack. In this category, a hardware device stores a value that both processes can view. One process then modifies the value to reflect the data being transmitted, while the other process reads the data. An attack model with more privileges for this channel can assume that the receiver is also able to modify the

TABLE I
CLASSIFICATION OF COVERT CHANNEL ATTACKS.

Hardware	Storage	Value	Section III-B
		Transition	Section III-B
	Timing	Value	Section III-C
		Transition	[12], [18]
Operating System	Storage	Value	[5], [15]
		Transition	Section III-D
	Timing	Value	Section III-E
		Transition	[15]
Network	Storage	Value	[1], [11]
		Transition	
	Timing	Value	
		Transition	[7], [16]

hardware value to reflect that the read is successful, thus providing a feedback channel and eliminating noise. A transition-based variant of this attack is possible in a more restrictive attack model. If the sender is only given the option to modify the value, a transition-based attack can occur by either modifying the hardware register to transmit a 1, or not modifying it to transmit a 0.

Our hardware, storage attacks use the PCI configuration space that is required to be implemented by every PCI-compliant device. When the system is initialized, the BIOS or operating system (or both), iterates through the PCI bus attempting to detect devices by reading their configuration registers. These registers contain several base address registers, which are eventually set by the OS to the memory ranges that the PCI device should respond to (which is how plug-and-play is supported). One of the registers is an address for the location of the expansion ROM, which contains initialization code for the specific PCI peripheral. Since this register is only useful when we are initializing the device, we reuse it for our own purposes to implement a CC[1].

The receiver proceeds by scanning PCI devices looking for a device that implements the Expansion ROM Base Address Register. The first such device that is detected is used in the attack. The size of the base address is variable depending on the amount of space used by the particular device. Thus, the first thing that happens is the length of the register is determined by writing all 1's to the register and then reading a value back. In our implementation, the Ethernet card was detected as implementing the expansion ROM base address register and it provided 16 configurable bits. After the receiver finds a register of appropriate size (at least 3 bits), a known value is written to signal the sending process. The sending process then begins and scans the PCI bus looking for the known value. As soon as it is detected, the sender begins sending data through the register.

This particular attack is a hardware-based, storage attack. Two variants of this are possible, value-based and transition-based. The key difference here is the threat model. While it is easier to transmit information by changing the value directly, an attacker may only have the ability to change the value to some other one (perhaps because he can only access the register through a limiting system call). In this scenario, a transition-based attack remains possible.

[1]Our system remained fully functional after changing this register.

C. Hardware Timing Value-Based Attack

In this type of channel, the sender is able to invoke the receiver at any time, but does so based on the value of a commonly observed hardware-based quantity. In this sense, the value present in the hardware at a particular time is of importance. If the hardware-based value is predictably increasing, it may be divided by a predetermined amount in order to reduce noise at the expense of transfer rate. This type of trade-off is present in several of our attacks and is analyzed in more detail in Section IV.

The threat model for this attack is slightly less restrictive than the general threat model. The sender does not need "write" access to any hardware value for this attack to work. The only privileges required is for both sender and receiver to have "read" access to a common value. It is further required for the sender to have "execute" or invoke rights over the receiver. This ensures that it can invoke the receiver whenever the common hardware location assumes the desired value.

For this attack we use one of the most quickly-changing values within a computer system: the internal processor cycle count. In our implementation we used a Pentium processor so we were able to access this value by issuing an RDTSC instruction. Notice that this register is not exactly a timer because the power settings can change the frequency of the processor and therefore the rate of increase in the cycle count.

D. Operating System Storage Transition-Based Attack

The next attack involves communication through the operating system. We assume two processes are running at different security levels and the sender can modify the view of the system presented to the receiver. This is a weaker assumption than arbitrary modification, and thus requires less privileges. Since the sender can not change the state to arbitrary values, it is the change in state, the transition, that is used to transmit information.

One form of this attack uses the file system. For instance, the sender can change or not change a property of a file to transmit 1 or 0. The receiver in this case does not need access to the file itself. It rather needs a mechanism to read that specific property; for example, by directory listing.

To transmit a bit covertly, the sender either changes or not changes one of the properties of a file (in our case

```
for each current_bit in data {
  while (true){
    milliseconds := current_time();
    if (current_bit ==
        (milliseconds / INT) mod 2) {
      invoke_receiver();
      break;}}}
```

Fig. 1. Algorithm for sender with operating system timing value-based attack

the filename). Note that this is a transition-based attack because the actual bit is never stored in the filename. Rather it is the changing of the filename that conveys information. In our implementation, we have used 32 files to transmit 32 bits of data at a time. These files can be located all in the same directory or in many directories in the system to fade the footprint of the attack making detection more challenging. In each step, the sender renames the file to a random string to imply a "1" and does not rename it to imply a "0". The only invariant during the transmission is the order of the files so that the receiver maintains proper bit order. We have implemented this by prefixing filenames with an ordered list of random prefixes.

E. Operating System Timing Value-Based Attack

Operating system, timing, value-based attacks require the sending process to have the capability to invoke or otherwise signal to the receiver. The sender looks at the current value of a common operating system data structure (such as the current time), and depending on the value (e.g., the last bit of the time value is a 1 or 0) invokes the receiving process. The receiving process reads this system data structure and can detect the transmitted value.

In this attack neither the sender nor the receiver need "write" access to shared resources. Rather the sender monitors a specific data structure inside the operating system and whenever it detects an appropriate value it invokes the receiver process that also can read this shared data structure. Note that in many security models, a higher security level process is allowed to execute a lower security one.

To implement this attack, we use the system timer as the shared resource. It changes frequently and many processes have read access to it. To transmit a bit of data covertly, the sender uses odd timer values to transmit a "1" and even timer values to transmit a "0". We also include the ability to adjust the frequency of the timer by dividing the time by a fixed value (INT). The algorithm for the sender is shown in Figure 1. This channel is noisy since the value of the timer may change before the receiver reads the value, thus causing an incorrect bit value.

F. Hybrid Covert Channels

So far, our attacks have focused on a single category within each of the three main classes. However, cc transmission is a two step process, with one optional step. First, the sending process performs some manipulation of a resource to send the data. Next, the receiver must receive the data by making an observation. Finally, the receiver can optionally send feedback to the sender, in an effort to both reduce noise or increase bandwidth.

We now present a new class of CCs, *hybrid*, in which the mechanism used for sending and receiving (and feedback) belongs to different categories. For example, an attack may send information by sending packets within certain time intervals, and the receiver may receive it by observing the state of the operating system.

Hybrid channels use an indirect flow of information between the different classification domains, thus even a proactive system administrator may not be able to reason about all the effects that occur when actions take place within a system. For example, even if strict firewall rules are in place such that the receiving process has no way to access the network data, unsolicited network packets still change the operating system state.

G. Hybrid Covert Channel Design and Implementation

In our hybrid CC, the sending occurs on a network, timing, transition-based channel, while the receiving is done through an OS, storage, transition-based channel. In this way the attack is fundamentally different from the previous designs and to the best of our knowledge, from all previous attacks discussed in literature.

The hybrid CC implemented, exploits a feature in Linux networking and the `/proc` filesystem. Namely, when a packet is sent to a machine, even if the corresponding port on that machine is closed and the packet is dropped, the OS packet counter increases. As a result, a sender sends a dummy packet to a closed port on the machine; the packet is dropped because there is no daemon to receive the packet; however, the data structure (the counter) holding the packet count in the kernel is incremented by one. Consequently, the receiver can look at this counter to conceive whether a packet has been sent or not (hence, receiving 1 or 0.)

This hybrid channel is intrinsically noisy. Some of the noise comes from the normal network traffic of the host and the rest of it is because of slight synchronization mismatch between the sender and receiver.

IV. Evaluation

We are interested in evaluating the maximum rate achievable for each channel and the amount of intrinsic noise. For noisy CCs, in order to evaluate the upper bound on the amount of information that can be reliably transmitted, we study the channel capacity. In all of the

channels studied, the noise comes either from the loss of synchronization or premature change of bit before the receiver gets the chance to read it. In any case, the probability of a bit flipping from 1 to 0 is the same as 0 to 1. In information theory, this is called a binary symmetric channel (BSC) [8]. The channel capacity, C for a BSC is given by:

$$C(r,p) = r \times (1 - H(p)) \quad (1)$$

$$H(p) = -p \times log_2(p) - (1-p) \times log_2(1-p) \quad (2)$$

where r is the channel rate (bits/time) and p is the probability of error which we calculate by the number of error bits divided by the total number of bits sent.

Our CCs are implemented on a computer with an Intel Core 2 Duo 3.00 GHz CPU and 2 gigabytes of RAM. We were running Fedora Linux with kernel version 2.6.24.7. The machine also contains an integrated Broadcom 5751 Gigabit Ethernet adapter. The results in this section are obtained without any attempt to optimize the implementation. The channel capacity is measured, in each case, by sending a large file through the channel, timing the transfer, and counting the error bits.

A. Hardware Storage Value/Transition-Based Attack

To evaluate the two hardware storage CC categories, we have implemented the Expansion ROM Base Address Register attack. Data is sent 14 bits at a time, with the least significant two bits being used for control.

After the sender writes new data, the least significant bit is set to a 0. After the receiver sees this event, he reads the data and sets the least significant bit to a 1. The next 14 bits are then sent. We measured the average channel capacity over 10 runs to be 2312 bits per second with a standard deviation of 534 bps.

B. Hardware Timing Value-Based Attack

The implementation of this attack deploys the hardware counter in the Pentium processors which can be accessed through RDTSC instruction. Since this counter changes much faster than the time it takes to invoke the receiver, we divide the 64-bit value by various divisors in order to slow down the apparent changing of the cycle count. This creates a trade-off between the bit rate and the amount of noise that is experienced. We evaluated this trade-off and illustrate the noise and the channel capacity versus bit rate in Figure 2. The maximum capacity we achieved was around 16 bits per second.

C. Operating System Storage Transition-Based Attack

We have implemented this attack using 33 files in Linux. 32 of the files are used for communicating data and one file is used as a feedback mechanism from receiver to the sender. The length of prefix in our implementation is two characters followed by five random characters.

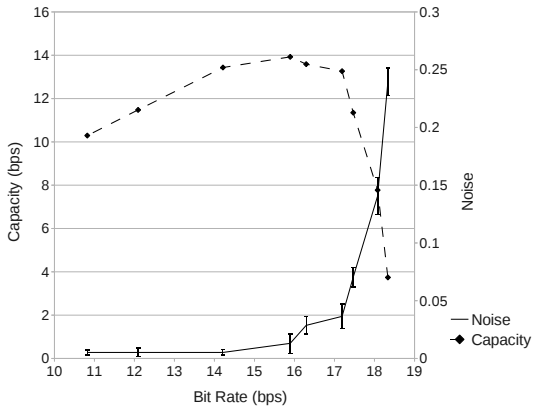

Fig. 2. The noise and channel capacity of the processor cycle-count attack.

This covert channel is almost noiseless, because it is highly unlikely that another process changes the name of the dummy files by chance. To evaluate the bandwidth of this attack, random strings of 10,000 bits has been sent through the channel in 10 different experiments. The average bit rate achieved for these experiments is 6721 bps and the standard deviation is 643 bps. Bit rate variations were caused by the overhead noise of other processes running on the machine. Since the channel itself is noiseless the capacity of the channel is the same as the bit rate.

D. Operating System Timing Value-Based Attack

The operating system, timing, value-based channel is implemented using the system timer in Linux. The noise and capacity of this channel is evaluated for 10 different bit rates. For each bit rate, 10,000 bits of data has been transferred through the channel (one bit at a time) and the experiment has been repeated 10 times (for a total 100 experiments). Figure 3 illustrates the amount of noise and channel capacity versus bit rate. Trivially, by shortening the interval (hence increasing the bit rate) the amount of noise grows quickly. For very low rates, the noise is low, but the capacity is limited by the bit rate. By increasing the bit rate, the capacity increases until it peaks at around 105 bps. After that point, the noise rises quickly resulting in a sharp decline in the channel capacity. A smart attacker tries to use an interval value which maximizes the capacity.

E. Hybrid Covert Channel Attack

We have implemented the network, timing, transition-based and operating system, storage, transition-based hybrid CC and evaluated its noise and capacity for 12

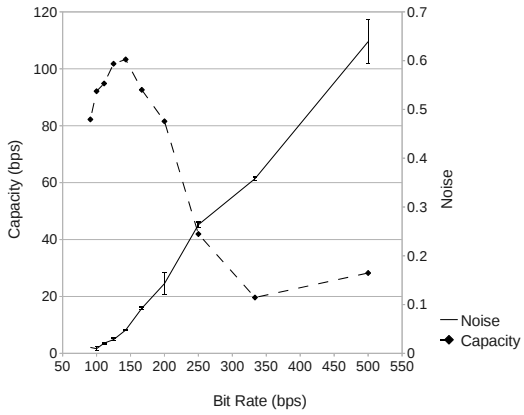

Fig. 3. The noise and channel capacity of the operating system timing value-based attack.

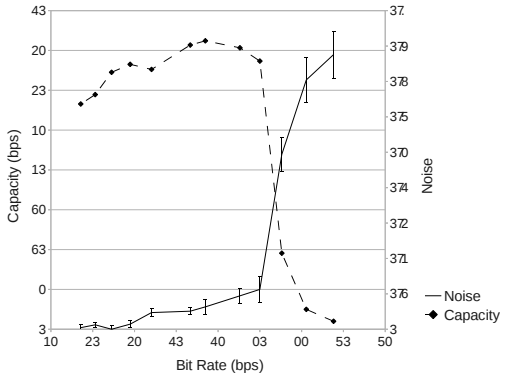

Fig. 4. The noise and channel capacity of the hybrid attack.

different bit rates. For each rate, 1,000 bits of data is sent through the channel (one bit at a time) and each experiment is repeated 10 times. The amount of noise and channel capacity versus bit rate is illustrated in Figure 4. For small bit rates, the amount of noise is low while it rapidly grows for bit rate higher than 50 bps. This approximately corresponds to the time required for receiving dummy packets and for the operating system counter to reflect the correct value (about 20 ms.) Beyond 50 bps, the sender and receiver quickly go out of sync since the counter reflects an old value for many of the time slots. The experiments are done under low normal traffic. The trend for channel capacity is the same as other noisy channels: the capacity is low for low bit rates, it peaks at some medium bit rate, and then it declines quickly for high bit rates because of high amount of noise. The maximum capacity of 32 bps is achieved for the bit rate of 35 bps.

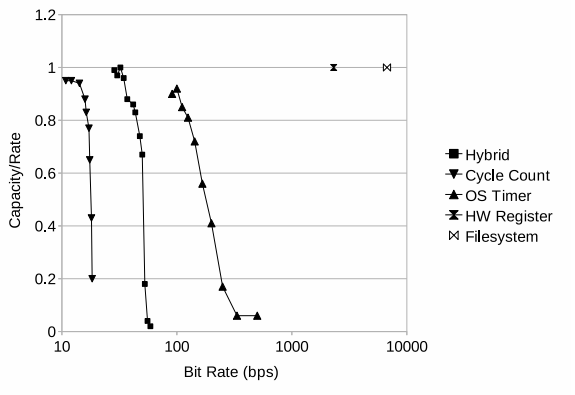

Fig. 5. Normalized capacity of various covert channel classes

F. Comparison

In order to more clearly compare different classes of covert channels, their capacity and bit rate are plotted in a single graph (Figure 5). To be able to compare their noise level, we have normalized their capacities by their bit rates. Hence, the y-axis of Figure 5 shows (1-H(p)) or the portion of bandwidth which can be used to transfer information.

The two noiseless channels (the hardware registers and the file system) are just single points in the graph. Noisy channels, on the other hand, have a trend capacity versus bit rate. The practical differences between these covert channels are apparent in the graph. As an example, for capacities near 1, the bit rate achievable by different channels vary from a few bits per second to 10,000 bits per second. A range of approximately three orders of magnitude.

V. Lessons Learned and Countermeasures

Throughout the implementation and evaluation process, several CCs properties have become apparent which we believe to be of value for protecting CIKR systems.

Each particular attack we designed is retroactively preventable using different defensive mechanisms. However, since CCs hide from existence, this retroactive response is weak. Even if one iterates all published CCs and prevents their communication (a difficult task), the system is unlikely to be CC-free.

One important observation is that each of our attacks uses mechanism-specific means to communicate. For instance, our hardware attacks use properties of the hardware (Pentium processors having a RDTSC instruction). The implications of this specificity for countermeasure development are important. If users are able to view and modify the details of individual pieces of hardware, for

example, then each of these hardware designs must be analyzed individually for CC vulnerabilities. Additionally, this heuristic can tell us where CCs are more likely to occur within CIKR systems.

Another lesson learned in this work is the dramatic differences between various CCs we studied. While in theory operating system-based storage and timing channels may look similar, one is able to achieve capacities of a few kilobytes per second (storage) while the other CC's capacity is about 100 bps. This means the threat model related to the specific environment must be fully understood and the threshold of acceptable amount of covert data must be known before one chooses the classes of CCs that may threaten the environment. For instance, sophisticated CCs with low channel capacity may be ignored for large scale control systems while they are hazardous for intelligence systems.

The concept of hybrid CCs contradicts the naïve belief that if all accesses from a process to a resource are blocked, the process cannot use that resource for a CC attack. For a system to eliminate the possibility of CCs using a resource, *every single* interaction of any entity (even other system components) with that resource must be fully understood.

VI. Related Work

CC related efforts can be divided into four major categories. Some of them propose new CC attacks and study their properties. IP time-to-live [11], MAC splitting [7], HTTP-based [1], and key-stroke-based [12] attacks are examples of new CCs. Cabuk [2] also designs new network-based CC attacks. Wang and Lee [18] propose new hardware-based CCs.

Another category analyzes known covert channel attacks usually using information theoretic techniques to study their properties such as capacity, error rate, or other channel-specific properties. Works by Venkatraman and Newman-Wolfe [14] and Moskowitz, et. al. [9] fall into this category.

The third category gives an overview of the covert channel attacks and the state-of-the-art [2].

Finally, the last category proposes countermeasures against CC attacks. Shared Resource Matrix (SRM) [6], information flow analysis [13], Covert Flow Tree (CFT) [5], time-domain anomaly [15], entropy-based techniques [3], and network pump [4] are among the proposed countermeasures.

VII. Conclusion

In this paper, we have focused on previously unexplored classes of CCs and designed attacks in each category. Furthermore, we have implemented and evaluated all of the designed attacks in real systems, presenting their threat model, bit rate, noise, and channel capacity. We believe that to protect CIKR systems against CCs, one must have an in-depth understanding of the practical differences between various classes of such channels as well as their threat models and potentials. This work is a step forward in understanding real-world properties of CCs and their differences.

We plan to study hybrid covert channels in more depth and provide implementations which incorporate coding and spread spectrum techniques. Future efforts can also focus on design and implementation of countermeasures against each of the covert channel categories using the lessons learned in this project and evaluate their effectiveness in various operating conditions.

References

[1] M. Bauer. New covert channels in http: Adding unwitting web browsers to anonymity sets. In *In Proceedings of the Workshop on Privacy in the Electronic Society*, pages 72–78, 2003.

[2] S. Cabuk. *Network Covert Channels: Design, Analysis, Detection, and Elimination*. PhD thesis, Purdue University, 12 2006.

[3] S. Gianvecchio and H. Wang. Detecting covert timing channels: an entropy-based approach. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 307–316, 2007.

[4] M. H. Kang, I. S. Moskowitz, and D. C. Lee. A network pump. *IEEE Transactions on Software Engineering*, 22:329–338, 1996.

[5] R. A. Kemmerer and P. A. Porras. Covert flow trees: A visual approach to analyzing covert storage channels. *IEEE Trans. Softw. Eng.*, 17(11):1166–1185, 1991.

[6] R. A. Kemmerer and T. Taylor. A modular covert channel analysis methodology for trusted dg/ux. In *Proceedings of ACSAC'96*, 1996.

[7] S. Li and A. Ephremides. A covert channel in mac protocols based on splitting algorithms. *Wireless Communications and Networking Conference, 2005 IEEE*, 2:1168–1173 Vol. 2, 2005.

[8] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[9] I. S. Moskowitz, R. E. Newman, D. P. Crepeau, and A. R. Miller. Covert channels and anonymizing networks. In *In Workshop on Privacy in the Electronic Society*, pages 79–88. ACM, 2003.

[10] D. of Defense. Department of defense trusted computer system evaluation criteria, 1985.

[11] H. Qu, P. Su, and D. Feng. A typical noisy covert channel in the ip protocol. *Security Technology, 2004. 38th Annual 2004 International Carnahan Conference on*, pages 189–192, 2004.

[12] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 5–5, 2006.

[13] C.-R. Tsai, V. Gligor, and C. Chandersekaran. On the identification of covert storage channels in secure systems. *Software Engineering, IEEE Transactions on*, 16(6):569–580, Jun 1990.

[14] B. R. Venkatraman and R. E. Newman-Wolfe. Capacity estimation and auditability of network covert channels. In *SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy*, page 186, 1995.

[15] C. Wang and S. Ju. Searching covert channels by identifying malicious subjects in the time domain. *Proceedings of the 5th Annual Information Assurance Workshop*, pages 68–73, 2004.

[16] Z. Wang, J. Deng, and R. Lee. Mutual anonymous communications: A new covert channel based on splitting tree mac. *INFOCOM'09: Proceedings 26th IEEE International Conference on Computer Communications*, pages 2531–2535, 2007.

[17] Z. Wang and R. B. Lee. New constructive approach to covert channel modeling and channel capacity estimation. In *ISC*, pages 498–505, 2005.

[18] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *Proceedings of ACSAC '06*, pages 473–482, 2006.