Exploring Covert Channel in Android Platform

Wade Gasior
University of Tennessee at Chattanooga
Chattanooga, TN USA

Li Yang
University of Tennessee at Chattanooga
Chattanooga, TN USA

***Abstract* — Network covert channels are used to exfiltrate information from a secured environment in a way that is extremely difficult to detect or prevent. These secret channels have been identified as an important security threat to governments and the private sector, and several research efforts have focused on the design, detection, and prevention of such channels in enterprise-type environments.**

Mobile devices have become a ubiquitous computing platform, and are storing or have access to an increasingly large amount of sensitive information. As such, these devices have become prime targets of attackers who desire access to this information. We explore the implementation of network covert channels on the Google Android mobile platform. Our work shows that covert communication channels can be successfully implemented on the Android platform to allow data to be leaked from these devices in a covert manner.

***Index Terms*—security, covert channel, mobile, Android**

I. Introduction

The Android platform accounts for just under 50% of the worldwide smart phone market [1] and combined with its open application market policy, is a prime target for malicious applications that steal users' data. Mobile platforms currently have only limited implementations of firewalls, intrusion detection systems, and other network security features, but this is likely to change as the information value that these devices hold increases.

Gaining a better understanding and developing improved methods of network covert channel prevention and detection are vital to the information security efforts in both private and government sectors, and have been the focus of much research in the past years. It is important to explore covert channel in mobile platforms to develop proactive protection and prevention mechanisms.

II. Background And Related Work

A. Purpose of Covert Channels

Covert channels can be described using the analogy of two prisoners attempting to escape. Simmons proposed this "prisoner problem" in 1983, which is the standard model used when describing covert channel communication [2]. The model describes two individuals, Alice and Bob, who are imprisoned and intend to escape. The two prisoners are allowed to speak with one another, but a third party (Wendy the Warden) monitors all communication between the two. In order to coordinate an escape plan, Alice and Bob must communicate with one another in a manner that does not alert Wendy, who will place the two in solitary confinement the moment she detects anything suspicious, making the escape impossible. In order to not be detected by Wendy, Alice and Bob must communicate messages that appear innocent, but contain hidden information that Wendy will not notice.

The primary goal of a covert channel is to hide the fact that communication is taking place at all. Covert channels differ from cryptography, where the primary goal is to transfer data that is only readable by the receiver [3] rather than hide the existence of communication. Covert Channels are similar to steganography, where a secret message is hidden or embedded within legitimate data, but are differentiated by the techniques used to hide the secret message. For example, a steganography approach might be to embed a secret message in the unused header fields of a TCP/IP packet, whereas a covert channel approach would be to encode the secret message by altering the delays between the TCP/IP packets.

Covert channels are desirable to exfiltrate sensitive information for several reasons. One, the use of a normal communications channel (such as an FTP or HTTP connection) is easily detected by wardens looking for malicious traffic. This type of traffic can be captured in log files and traffic dumps, and then analyzed and prevented. Making the communication channel more obscure, by methods such as using nonstandard port numbers, is also easily detectable and would trigger mechanisms such as packet anomaly detection systems [3].

Our goal in this work is to show that communication channels between an Android device and a remote server can be implemented in ways that are undetectable by network wardens.

B. Types and Classifications of Covert Channels

Covert channels can be employed in a number of scenarios where data needs to be transferred undetected. For example, imagine that an attacker has compromised a system within a secure computing environment, such as a financial institution or military base, and gained access to sensitive information. These types of environments employ a variety of network security features such as firewalls and intrusion detection systems to detect and prevent the leak of such sensitive data to outside networks or systems. The challenge for the attacker is to exfiltrate this data to an unsecure location without being detected, and covert channels provide a means to do so.

Network covert channels can, in a broad sense, be classified as either storage-based (Figure 1) or timing-based (Figure 2), but the distinction between the two is quite blurred. *Storage-based* covert channels, or covert storage channels (CSC), operate by altering the content of some resource that can be observed by a receiver [4]. Cabuk describes the implementation of a simple binary CSC in [4]. This channel operates using a timeline divided into intervals of size t, known by both the sender and receiver. The sender transmits a bit value of 0 by maintaining silence throughout a given interval, and transmits a bit value of 1 by sending a packet or

becoming active during a given interval. *Timing-based* covert channels, or covert timing channels (CTC), operate by altering the delays between network events, such as the sending of a packet. Berk implemented one of the earlier examples of a network covert timing channel by encoding a message using interpacket delays [3]. This channel operates using a set of time intervals $t_1, t_2, \ldots, t_n$. Each time interval is associated with a symbol from the input alphabet, and the delay between consecutive packets (interpacket delay) is altered to transmit a given symbol.

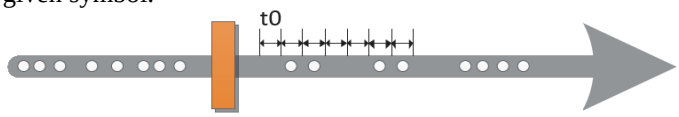

Figure 1 Storage Covert Channel

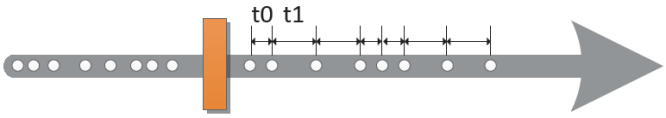

Figure 2 Timing Covert Channels

Network covert channels have been implemented in a number of ways, all designed with the goals of resistance to detection and increased capacity versus the basic timing and storage implementations described above. The technique of implementing a covert storage channel using packet rate modulation, discussed in [5], encodes information by establishing a time interval t and transmitting data during a given interval t_n at a rate representing a specific input symbol S_i. This results in a more robust covert storage channel compared to the basic implementation discussed in section 2.4. In a simple binary scenario, a sender could encode a binary 0 by transmitting at a rate r_0 for a period of length t, or a binary 1 by transmitting a rate r_1 for a period of length t. The technique of implementing a covert timing channel by replaying interpacket delays observed in legitimate traffic is discussed in [6]. The technique uses a sample of interpacket delays observed from legitimate traffic (S_{in}) partitioned into two equal bins S_0 and S_1. To transmit a binary 0, a random interpacket delay from S_0 is transmitted. To transmit a binary 1, a random interpacket delay from S_1 is transmitted. Model-based covert timing channels, discussed in [7] and [8], are implemented in a manner that mimics the statistical properties of legitimate traffic, making them very difficult to detect. Model-based CTC's are implemented in four phases. The first phase observes and records the interpacket delays of legitimate traffic. The second phase analyzes the traffic to determine the best distribution model (Exponential, Gamma, Pareto, Lognormal, Poisson, Weibull, etc.) by using root mean squared error and maximum likelihood estimation. Once a model has been chosen, the third and final phase determines the appropriate interpacket delay times to be used for the covert channel using the inverse distribution function of the selected model and encodes the channel. Decoding by the receiver is performed using the cumulative distribution function. A transmission control protocol (TCP) databurst is the number of TCP segments sent by a host before waiting for a TCP ACK packet. Luo, in [9] discusses the technique of implementing a covert storage channel by altering the size of these databursts.

III. Implementation of Network Covert Channels on the Android Platform

To evaluate covert channels on the Android platform, we designed two implementations that stealthily transmit data from the phone to a remote server: a timing-based covert channel and a storage-based covert channel. Our timing-based CC is encoded using delays between network events, while our storage-based CC is encoded based on the ordering of network events. Both implementations involve embedding a covert channel in an innocuous appearing application.

A. Timing-CC Implementation Overview

In order to implement a timing-based channel on the Android platform, we required an application that would generate a large amount of legitimate traffic at a steady rate in order to make the embedding of a covert timing channel effective. To accomplish this, we developed an application that transmits live video from the camera on the Android device to a remote server (presumably controlled by the attacker). In a real-world implementation, this server could then broadcast the video over the Internet similar to other applications such as Justin.TV®. The application essentially allows the phone to operate as an IP camera, and gives us plenty of overt traffic in which to embed a covert channel. The protocol we designed for this application sends one video frame per TCP message. The first four-byte segment of each TCP payload contains an integer with the size in bytes of the image payload.

In order to implement our covert channel, we alter the delays between sequential TCP messages. Delays are introduced between TCP messages on the Android device by using calls to the sleep method in the Android *SystemClock* library between TCP message transmissions. Our network covert timing channel uses binary encoding to transmit messages. In order for the covert channel to signal to the server that it is ready to begin transmission, it transmits the sequence of binary symbols 0-1-0-1-1-0-1-1 (Ascii code for '['). To signify the end of a transmission, the application simply stops transmitting any input symbols that represent a binary 1.

The server-side of this application displays video streamed from the mobile device while capturing and recording the delays between received messages. Delays on the server-side are calculated by measuring the time between receipts of complete TCP messages. This is done by using calls to the *nanoTime* method in the Java System library. If a delay is above a certain threshold value, the observed delay is treated as a binary 1. Otherwise, it is treated as a binary 0. The application reconstructs the transmitted covert message and displays it in plain text as it is received.

B. Storage-CC Implementation Overview

To implement a storage-based covert channel, we designed an application that displays a small advertisement banner at the bottom of an arbitrary application. The advertisements are

fetched from our remote server, of which there are n choices of advertisements.

The application leaks information by requesting a specific advertisement to represent a specific encoded input token (binary values of the contacts list). If 2^n advertisements are available, then each request can represent n bits of the data to be transmitted. The application fetches advertisements using HTTP requests with a POST parameter representing the specific ad to be fetched.

Specific advertisements represent the beginning or end of a covert transmission. The server-side of this application responds to HTTP requests with the appropriate ad, and records the sequence in which ads are requested during a covert transmission. The application displays covert data in plain text as it is received by decoding the message based on the order of advertisement banners requested.

C. Implementation Challenges

Several challenges were faced during our research and development of covert channels on the Android platform. A challenge we faced during our research was that of accessing the targeted sensitive data on the Android device. The Android operating system uses fine-grained per-application permissions that the user must approve at install time (e.g. permission to access the network and permission to access the contacts list). Applications that have both network access and access to sensitive data such as the contacts list raise suspicions from the user who must approve these permissions, and from security software that identifies over-privileged applications [11]. Exasperating this problem is the fact that applications on the Android platform are executed in isolated runtime environments. The Android operating system is a Linux-based OS where each application is run as a distinct user and group, which creates a sandboxed environment that separates each application from one another and from the underlying system. This prevents using the approach of dividing the attack into two separate applications - one for the data access and one for the network connectivity - because communication between the two is not possible.

A second challenge faced during our research involved the network implementation of our covert channels. The Android platform (and the Java programming language in general without accessing native libraries) does not allow an application to have access to raw sockets, so techniques used in traditional computing environments that alter the flow of packets in order to implement covert channels [3, 5, 6, 10] are not possible. Network programming on Android is limited to high-level socket communications (transport layer), and does not provide the ability for programmers to access lower levels of the network stack. The presence of low level socket access would represent a major security risk, as these types of operations require root access on the device.

A third challenge faced during our research was the difficulty of implementing our timing-based covert channel over the cellular network. The mobile phone network introduced a large amount of jitter and highly varying quality that caused synchronization and reliability issues in our implementation. We also discovered that very few UDP packets complete the trip from source to destination without being dropped.

D. Solutions of Challenges

To overcome the challenge of accessing sensitive data on the Android platform, we relied on the method of on-device covert storage channels discussed by Schlegel et al. [11]. Schlegel showed that the protection mechanisms on the Android platform that prevent unauthorized application-to-application communication can be subverted. The implementation puts in place malware in the form of two cooperating applications: one with permission to access sensitive data, and the other with network access. On-device covert channels were used to communicate data between the two applications. These channels rely primarily on changing a global setting that both applications had access to, such as the device's volume or screen brightness.

Based on Schlegel's findings, we assume that our covert channel applications have access to the desired sensitive data on the target device while also having full network access, as would be the case for a real-world application. To overcome the challenge of the lack of low level socket access, we implemented custom TCP and UDP protocols to carry our legitimate application traffic. We encode a covert timing channel within the legitimate channel by altering the delays between protocol messages. We insert enough of a delay between protocol messages that the TCP stream is flushed between each message. We also disable TCP's Nagle algorithm, a congestion control technique used by TCP to combine multiple messages into a single packet to reduce the number of packets sent over the network. This allows us to have finer grained control over the timing of TCP messages. The traffic quality challenges presented by the cellular network have forced us to use the TCP protocol only, since UDP traffic often does not make it through the network. We also used higher values for inter-message delays than would be necessary if access to raw sockets existed. This resulted in a lower bandwidth covert channel than one would normally be able to implement.

E. Experimental Setup

For our mobile platform, we used a Motorola Droid X Smartphone. This platform was chosen because it is a fairly common smartphone, with widespread use in the real-world. The Droid X phone is powered by a 1GHz Texas Instruments Open Multimedia Application Platform (OMAP) 3630 processor chip, which uses an ARM Cortex-A8 processing unit. The unit has 512MB of random access memory (RAM). The Droid X used during our experiments was running Android version 2.3.3 (codenamed Gingerbread) as its operating system. This was the most current operating system for this device as of the time of our research. For our experiments, all other running processes on the phone were killed other than our covert channel application and required system processes.

For our server platform, which served as the communications endpoint for both overt and covert traffic generated by our applications, we used a system powered by an Intel Core i7 processor with 12GB of RAM running

Windows 7 64-bit. The Droid X used during our experiments was connected to the Internet via Verizon's 3G service in Chattanooga, TN. Verizon's network type in this location is a Code Division Multiple Access (CDMA) Evolution-Data Optimized (EV-DO) revision A network. Our typical signal strength during testing was -69 dBm, and all tests were performed from the same location. As a reference, signal strength of -60 dBm represents a nearly perfect connection, while signal strength of -112 dBm represents a nearly unusable connection.

We performed several tests to establish a measure of network quality. We used the SpeedTest.net Android application by OOKLA in order to get a general idea of the quality of the Droid X's Internet connection. This application tests the uplink and downlink speed to a geographically close server. We used the application to test connectivity to a server in McMinnville, TN hosted by Ben Lomand Telephone Systems. We performed multiple tests on weekday evenings (approximately 7:00pm local time) to capture network conditions during a time of relatively heavy network load. The speed tests provided fairly consistent results: approximately 2.0 Mbps downlink speed; approximately 0.7 Mbps uplink speed; and a ping time of approximately 200 ms.

For our covert timing channel, we implemented a basic binary channel (two input symbols: binary 0 and binary 1). We designed two Android applications: a testing application that implements a covert channel without any overt traffic, and a real-world application that embeds a covert channel within overt traffic. The first was designed only for the purpose of determining baseline measurements of the network factors that would affect our covert channel, such as the correlation between transmitted symbols and observed symbols, the jitter introduced by the cellular network, and the max throughput capable for a binary channel. The channel is implemented by encoding covert traffic using delays between 1-byte TCP messages.

In order to establish baseline measurements of jitter, input to output symbol accuracy, and max throughput, we used our implementation that produces only covert traffic (no overt traffic) to perform experiments to test these factors. To test jitter and input to output symbol accuracy, we used the application to transmit several various input symbols (various delays between TCP messages) 350 times each, and measured the observed symbol (observed delay) at the server end. These tests were all performed under similar network conditions. The goal of this experiment was to find a suitable input symbol to represent the long delay in our binary input alphabet (the short delay is represented by transmitting messages with no delay).

Next, to test channel throughput and accuracy, we transmitted a short message ("wadegasior@gmail.com") encoded in binary multiple times using various delays and measured the rate and accuracy of the message being received at the server-side. When transmitting string messages using our covert timing channel, each character of the string is transmitted using eight bits represented by the character's binary ASCII value.

We then performed similar experiments using the second application, which embeds a covert channel within an overt video stream. These experiments were performed to determine the amount of jitter caused by the additional overt traffic on our covert channel, the accuracy of our covert channel in a real-world scenario, and the maximum bandwidth of our covert channel in a real-world scenario.

Our covert storage channel is implemented using advertisement banners fetched from our server and displayed within the application on the Android device. We experimented with various input-output alphabets (number of advertisements available to fetch) and various delays between fetches to determine the throughput of this covert channel implementation. The size of the input-output alphabet determines the amount of information that can be transmitted with each advertisement request. In order to transmit n bits with each request, 2^n advertisement banners must be available to fetch. The delay between advertisement banner fetches is governed by what a typical user would expect as a normal rate at which an advertisement banner would be updated. For example, if our embedded advertisement banner updates once every three seconds, it would be a telltale sign that suspicious activity is going on.

IV. Experimental Results and Analysis

Our first experiment uses a covert timing channel with no overt traffic overhead. We transmitted the delays listed in Table 1 350 times each, and analyzed the accuracy at which these delays were observed by our server. Our goal was to find the smallest input symbol (smallest inter-message delay value) that could be clearly distinguished from an input symbol of 0 ms.

TABLE I
Input Delays Transmitted During Baseline Covert Timing Channel

Delay (ms)
0
25
50
75
100
150
200

First, we transmitted TCP messages with an inter-message delay of 0 ms (no delay). The delays observed at the server-side showed a saw tooth pattern, where delays alternated between relatively high values (80 to 100 ms) and a value of 0 ms. Measurements for the first twenty observed delays are shown in Figure 3.

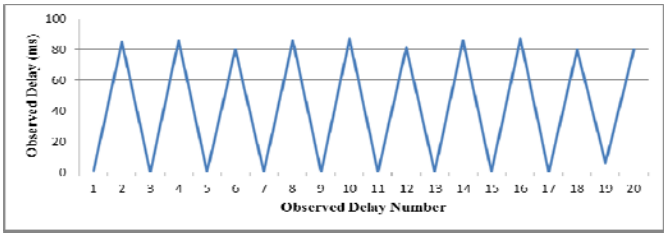

Figure 3 Output Symbols Observed with Input Symbol of 0 ms
(First 20 Observed Delays)

The observed saw tooth pattern can be attributed to the delayed ACK feature present in the TCP protocol. TCP does not immediately respond to every received TCP packet with an ACK. TCP waits for a period, expecting that an application response might be sent, and the ACK can be included in the response to save bandwidth. If a second TCP packet is received during this waiting period, TCP immediately responds with an ACK. Because our testing application was transmitting TCP messages consisting only of a single packet, the server waited to respond with an ACK after every other packet causing this result. We verified this behavior by analyzing the traffic with Wireshark. Our Android application sends two packets, and then waits until it receives an ACK before sending more, and thus every other delay is a high value. Taking this into consideration, we redesigned our server application by having it respond to every TCP message received with a short response. This effectively sends an immediate ACK for every message received. Repeating the experiment with this change yielded the observed values shown in Figure 4, again for the first twenty delays observed.

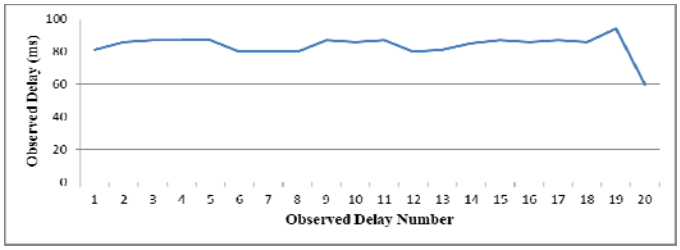

Figure 4 Output Symbols Observed with Input Symbol of 0 ms, with forced ACK after each message (First 20 Observed Delays)

With this modification, we were able to obtain a much more consistent input symbol to output symbol relationship. For our sample size of 350 delays, we observed a mean delay of 88 ms, a range from 60 to 127 ms, and a standard deviation (jitter) of 10 ms. A frequency of observed symbols is shown in Figure 5.

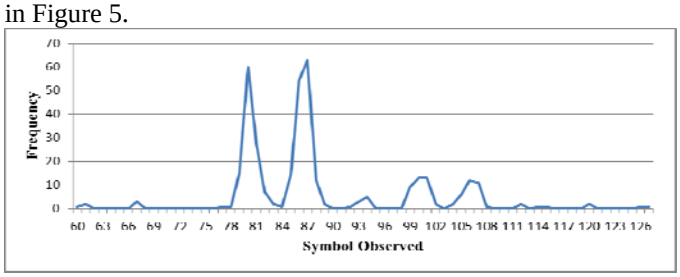

Figure 5 Output Symbols Observed by Server with Input Symbol of 0 ms (350 samples)

After determining possible input symbol values for representing a binary 1 for our covert channel, we tested the accuracy and overall throughput of channels implemented using these input symbols. Again, these channels were implemented without the overhead of overt traffic. For each combination of input values, a threshold value was selected based on the results of the previous experiments. If a delay is observed below the threshold value, it is treated as a binary 0. If a delay is observed that is greater than the threshold value, it is treated as a binary 1.

For each combination of input and threshold value, we measured the accuracy and throughput of a covert channel by transmitting the string value "wadegasior@gmail.com" encoded to 7-bit ASCII. The results are shown in Table II. Each combination was tested five times, and the average accuracy and throughput recorded.

TABLE II
BASELINE ACCURACY AND THROUGHPUT EXPERIMENT RESULTS

Input Symbol for Binary 0 (ms)	Input Symbol for Binary 1 (ms)	Threshold Value (ms)	Throughput (bps)	Accuracy (%)
0	100	95	9.3	81.7
0	150	110	7.8	99.2
0	200	130	7.2	100

These results demonstrated an expected correlation between delay length, accuracy and throughput. Using larger delays for the binary 1 input symbol (with a larger associated threshold value) results in more accurate transmissions at the cost of throughput.

IV. CONCLUSION

In this work, we implemented network covert channels on mobile devices that allow data to be leaked from the device via its network connection in a manner that is very difficult to detect. We proved this by implementing both timing-based and storage-based network covert channels on an Android-powered smartphone, and using these channels to covertly transfer information from the phone to an external server.

REFERENCES

[1] B. Reed (2011, Aug. 1). Android market share nears 50% worldwide [Online]. Available: http://www.networkworld.com/news/2011/080111-canalys.html

[2] G. J. Simmons, “The prisoners’ problem and the subliminal channel,” in Advances in Cryptology: Proceedings of Crypto '83, pp. 51-67, Plenum Press, 1984.

[3] V. Berk, A. Giani, and G. Cybenko, “Detection of Covert Channel Encoding in Network Packet Delays,” Tech. Rep. TR2005-536, Dartmouth College, Computer Science, Hanover, NH, August 2005.

[4] S. Cabuk, C. E. Brodley, and C. Shields, “Ip covert channel detection,” ACM Trans. Inf. Syst. Secur., vol. 12, pp. 22:1-22:29, April 2009.

[5] S. Zander, G. Armitage, and P. Branch, “A survey of covert channels and countermeasures in computer network protocols,” Communications Surveys Tutorials, IEEE, vol. 9, pp. 44 -57, 2007.

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

[7] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia, “Model-based covert timing channels: Automated modeling and evasion,” in Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, RAID ’08, (Berlin, Heidelberg), pp. 211-230, Springer-Verlag, 2008.

[8] S. H. Sellke, C.-C.Wang, S. Bagchi, and N. B. Shroff, “Tcp/ip timing channels: Theory to implementation.,” in INFOCOM, pp. 2204-2212, IEEE, 2009.

[9] X. Luo, E. Chan, and R. Chang, “Tcp covert timing channels: Design and detection.,” in *DSN*, pp. 420-429, IEEE Computer Society, 2008.

[10] S. Gianvecchio and H. Wang, “An entropy-based approach to detecting covert timing channels,” *IEEE Transactions on Dependable and Secure Computing*, vol. 99, 2010.

[11] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. *In Proceedings of the 18th Annual Network and Distributed System Security Symposium*, NDSS '11, pages 17-33, 2011.