Sensing Danger: Exploiting Sensors to Build Covert Channels

Thomas Ulz, Markus Feldbacher, Thomas Pieber and Christian Steger
Institute of Technical Informatics, Graz University of Technology, Graz, Austria

Keywords: Sensor, Exploit, Security, Side-channel, Covert Channel.

Abstract: Recent incidents have shown that sensor-equipped devices can be used by adversaries to perform malicious activities, such as spying on end-users or for industrial espionage. In this paper, we present a novel attack scenario that uses unsecured embedded sensors to build covert channels that can be used to bypass security mechanisms and transfer information between isolated processes. We present covert channels that require read- and write-access for sensor registers as well as a covert channel that transfers data by just triggering sensor readings so that malicious behavior cannot be distinguished from normal sensor usage. For each presented covert channel we discuss the trade-off between data rate and the likelihood of being detected as well as potential countermeasures. The fastest covert channel we implemented achieves a data rate of 4844 bit/s while the stealthiest but slower covert channel cannot be distinguished from normal user behavior. To highlight the significance of these security issues, we used popular platforms, such as Linux and Android, to evaluate the presented covert channels. However, we do not make any assumption regarding the device's platform, and thus we believe that the presented exploits pose a significant security risk for any sensor-equipped device.

1 INTRODUCTION

Nowadays, sensors are embedded into nearly every device to improve the device's usefulness. Applications of such sensor-equipped devices are basically unlimited and include, for example, environmental monitoring (Srbinovska et al., 2015), healthcare applications (Nguyen et al., 2016), or industrial applications (Chi et al., 2014). Also, modern smartphones contain multiple embedded sensors that are used to improve user experience (Yu et al., 2015). Regardless of the application domain, embedded sensors are seen as an enabling technology for improved functionality such as context awareness (Perera et al., 2014). However, including embedded sensors into everyday objects also entails several security risks. The most addressed security issue regarding sensors is the privacy aspect of sensor data (Suo et al., 2012). Since sensors observe the environment, they sense private information, such as health care data (Yi et al., 2016) or industrial processes (Sadeghi et al., 2015). A loss of such private sensor data can lead to severe consequences that can even result in severe financial losses for a business if intellectual property or customer data is lost in a security breach. Therefore, the privacy of sensor data usually is considered to be of high importance. The second security issue related to sensors is the trustworthiness of sensor data (Suo et al., 2012). In so-called deception attacks (Kwon et al., 2013), an adversary influences a system's behavior by manipulating sensor data. If the manipulated sensor data is used to control a system or a process, the system could be physically damaged or even threaten human lives due to its malicious behavior (Derler et al., 2012). Therefore, the trustworthiness of sensor data also is considered to be of high importance. Finally, insufficient and too coarse permissions for accessing sensors also present security issues in sensors that need to be addressed (Shrivastava et al., 2017). However, such issues most often are associated with privacy concerns. In this paper, we are going to exploit insufficiently secured sensor interfaces to transfer data between two processes that are otherwise prevented from exchanging data. A so-called covert channel poses an immense security risk for systems since the security implications range from leaking private information to compromising a system so that its intended behavior is either manipulated or disabled. We present three different covert channels that provide a trade-off in covert channel data rate and the likelihood of such a covert channel being detected by a user or some software mechanism such as auditing sensor usage (Mirzamohammadi et al., 2017). The data is transferred by exploiting unprotected sensor registers in all three presented approaches. We do

Ulz, T., Feldbacher, M., Pieber, T. and Steger, C.
Sensing Danger: Exploiting Sensors to Build Covert Channels.
DOI: 10.5220/0007353801000113
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 100-113
ISBN: 978-989-758-359-9

not claim that the list of covert channels presented in this paper is exhaustive. Instead, with this paper we want to bring attention to such security issues and highlight the importance of mitigating them.

Contributions. In brief, we make the following contributions in this paper. To the best of our knowledge, we are the first to present these concepts. We present three sensor-based covert channels that are enabled by unprotected registers in embedded sensors. These covert channels differ in the achievable data rate and the channel's likelihood of being detected. In addition, we present countermeasures to mitigate the presented covert channels. We also demonstrate a sensor-based covert channel that is based on exploiting a security weakness in Android's sensor management system. To facilitate the evaluation of sensors regarding exploitable vulnerabilities, we developed an easy-to-use modular and extendible framework. We provide this framework on GitHub[1].

Outline. The remainder of this paper is structured as follows. In Section 2, we are going to briefly introduce side-channels and covert channels, and categorize these attacks. We list other state-of-the-art covert channels in Section 3 and discuss their performance. After that, we define the underlying system-model we assume and discuss possible resulting threats in Section 4. In Section 5, we demonstrate our register-based covert channels and discuss potential countermeasures. Section 6 discusses covert channels based on exploiting Android's sensor manager. The framework we developed for evaluating sensor-based covert channels is then presented in Section 7. In Section 8, we evaluate the presented covert channels as well as our framework's functionality. This paper is then concluded with Section 9.

2 COVERT CHANNELS

The term covert channel was coined by Lampson (Lampson, 1973) in 1973 when he defined a covert channel as a communication channel that is *not intended* for information transfer at all. Usually, covert channels facilitate information transfer between processes that are otherwise prohibited from communicating with each other by the system. In order to build a covert channel, the data that needs to be transferred is embedded in events that are observable by other processes such as a processes's system load (Lampson, 1973). Such observable events are denoted as so-called *side-channels*.

[1]https://github.com/Grundplatte/SensIO

Figure 1: Basic concept of a covert channel.

Side-Channels: can be exploited for either *active* or *passive* side-channel attacks. In active side-channel attacks, an attacker actively tampers with the device, thus requiring physical access (Genkin et al., 2016). In passive side-channel attacks, the effects that are caused by one process are observed by another process. This can be used to reveal confidential information such as cryptographic keys by monitoring processes for *unintentionally* leaking side-channel information, such as timing, power consumption, or electromagnetic emanation (Kim and Quisquater, 2007; Longo et al., 2015; Luo et al., 2015). If a process *intentionally* triggers such observable effects, data can be transferred by the process and received by another process, thus establishing a so-called *covert channel*.

Covert Channels: in general comprise three entities, a *sender - receiver* pair, and the *side-channel* that is used to build the respective covert channel. Figure 1 illustrates a basic covert channel and the data flow between these entities. (i) The **sender** is in possession of data that it wants to transfer to the receiver. However, the system prevents the sender from using conventional methods, such as shared memory or sockets, to transfer its data. Therefore, the sender utilizes side-channel information that can be manipulated by the sender. The (ii) **side-channel** is used by the sender and receiver as a stealthy transport medium for their data transfer. (iii) The **receiver** needs to be capable of observing the side-channel's state changes. In addition, it must be synchronized with the sender so that the start and end of the transferred data stream can be correctly identified. In an ideal scenario, the receiver is also able to distinguish between state changes of the side-channel that are either caused by the sender or by normal system operation.

Depending on a side-channel's nature, different data rates can be achieved. The achievable speed depends on two factors. The first determining factor is the *component's speed*, i.e. sensor-based covert channels (Carrara and Adams, 2016) will be slower than covert channels based on components that are optimized for performance such as memory. The second determining factor is the *word size* that can be transferred. A covert channel that is capable of transmitting a multi-bit word per time unit will be faster than a covert channel that only can transfer a 1-bit word per

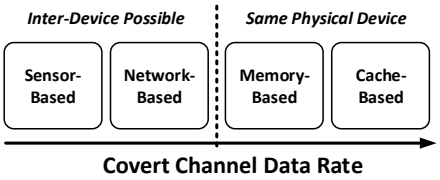


Figure 2: Classification of different covert channel types according to the covert channel data rate.

time unit. A basic overview that compares the achievable covert channel data rates for the most common side-channels is shown in Figure 2. We also classified the covert channels based on their *scope*. The faster covert channels (memory- and cache-based) require the sender and receiver to reside on the same physical device so that both processes share the same memory or cache. In contrast to that, sensor- and network-based covert channels can transfer data between processes that are running either on the same physical device or on different devices as long as they can access the *same* shared medium.

3 RELATED WORK

In this section, we list related work for exploiting different system components.

3.1 Cache-based Covert Channels

Modern processor- and system-architectures entail leaking side-channel information due to these systems being optimized for performance or energy efficiency (Wang and Lee, 2006). One of these side-channels that leak information is cache memory. Cache-based side-channels do not rely on weaknesses in the operating system (OS) or a virtual machine monitor, and thus, these attacks are considered to be highly practical (Liu et al., 2015). The side-channel that is exploited for cache-based covert channels is the timing difference between a cache *hit* and a cache *miss*. If a process is capable of intentionally causing cache hits or misses, data can be encoded in these events. A cache miss can be provoked by flushing all data from cache regions (Osvik et al., 2006; Yarom and Falkner, 2014; Gruss et al., 2016). A very fast and reliable cache-based covert channel that is capable of bit rates over 45 KByte/s with a bit error rate of 0% was presented in literature (Maurice et al., 2017).

3.2 Memory-based Covert Channels

Since the memory in modern processors and systems is shared between cores, memory-based side-channels are used to reveal confidential information and to build covert channels (Zhang et al., 2012). One commonly used method for inter-process communication, shared memory, is usually prohibited by process isolation, such as sandboxes or virtual machines. However, side-channel information can be used to bypass these protection mechanisms, for example by exploiting memory deduplication (Xiao et al., 2013). Other side-channels exploit timing differences while locking the memory bus (Wu et al., 2011). A DRAM-based side-channel was presented (Pessl et al., 2016) for which the authors claimed raw bit rates of up to 2 Mbit/s while the bit error probability stayed below 1%. However, the authors did not state a bandwidth for 0% bit errors.

3.3 Network-based Covert Channels

Exploiting network protocols to build network-based covert channels is one of the earliest known methods for stealthy data transfer. Network-based covert channels are used to bypass network protection mechanisms such as firewalls or virtual local area networks (VLANs) that otherwise are used to monitor or prevent unwanted data transfer (Zander et al., 2007a). To hide transferred data in network packets, various protocols at different network layers are exploited. On the network layer, information can be hidden in protocol headers such as in the 802.11 protocol's sequence control field (Frikha et al., 2008), or in the Received Signal Strength Indicator (RSSI) (Tuptuk and Hailes, 2015). On the Internet and Transport layer, many approaches use Transmission Control Protocol/Internet Protocol (TCP/IP) header fields to hide data in network packets (Ahsan and Kundur, 2002; Giffin et al., 2002; Zander et al., 2007b). On the application layer, various protocol fields can be used to hide information (Ameri and Johnson, 2017). Also, covert channels that work *independently* of any network protocol were presented (Cabuk et al., 2004; Ji et al., 2009).

3.4 Sensor-based Covert Channels

Malicious use of sensors and their data traditionally involves spying on events or humans to reveal confidential information (Perrig et al., 2004). If an event (e.g. entering a password) triggers physical effects such as vibration that can be measured by a nearby sensor, sensor-based side-channels can be used for malicious activities (Aviv et al., 2012). Also other sensors such as ambient light sensors can be used to steal confidential information on a smartphone (Spreitzer, 2014). In the same manner, sensor-based covert channels can be built by triggering physical effects

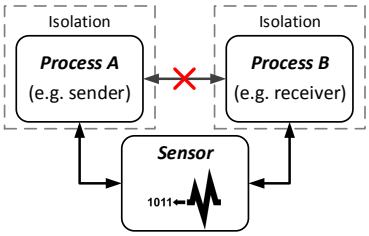


Figure 3: Underlying covert channel system-model.

from one process and by measuring these effects from another process. For instance, covert channels based on the temperature of devices were presented (Brouchier et al., 2009; Guri et al., 2015). In contrast to approaches that require the modification of physically observable values, covert channels might also tamper with measured sensor values for stealthy data transfer (Tuptuk and Hailes, 2015).

4 SYSTEM- & THREAT-MODEL

In this section, we present the system model that is used to exploit embedded sensors and discuss the threat-model and potential impact of covert channels.

4.1 System-model

To exploit security weaknesses in sensors for building sensor-based covert channels, we consider the system-model shown in Figure 3. This model is comprised of at least two potentially *isolated processes* and a *shared sensor*. The isolation between processes (e.g. sandboxes) prevents any direct data exchange between these processes. However, in our system-model both processes can access the *same* shared sensor. We do not make any assumption regarding the type of sensor that is present in this system.

4.2 Threat-model

In our threat-model, we identify two scenarios that are enabled by transferring data over a covert channel. An isolated process might be able to send private data or receive instructions via this covert channel.

1. We assume that an isolated *process A* holds confidential information that an attacker wants to communicate to another *process B*. For instance, *process A* might monitor a video stream to detected movements in a security system. However, *process A* is prohibited from sharing the video stream. If *process A* is capable of transferring information stealthily to *process B* using a covert channel, data privacy is broken.

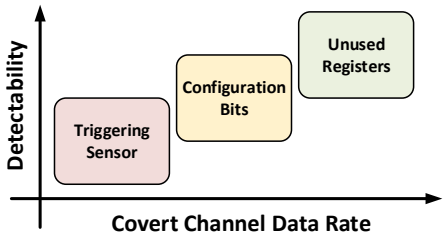


Figure 4: Trade-off between data rate and detectability of our presented covert channels.

2. As second scenario, we assume that *process A* is capable of controlling some physical process such as a robot's actuators. To prevent malicious control actions, *process A* is isolated from the network. Instead, control actions are solely triggered by sensor data. However, an attacker might be able to send control commands to *process A* via *process B* and a covert channel.

To successfully establish a sensor-based covert channel between two processes, an attacker needs to be capable of executing modified code in both involved processes. We assume that an attacker is capable of running the required malicious code through any state-of-the-art attack such as code injection (Poeplau et al., 2014).

5 SENSOR REGISTER EXPLOITS

In this section, we present three exploits of embedded sensors that we use to build sensor-based covert channels. All three exploits are based on direct access to the sensor. That is, access to the sensor is not limited by any mechanism such as managed sensor access (Milette and Stroud, 2012). In all three approaches, sensor registers are used to transfer information between processes. The approaches differ in the achievable covert channel data rate and the likelihood of detecting such a covert channel (*detectability*). Also, the required effort for mitigating the different covert channels differs. A comparison of all three approaches regarding these three attributes is shown in Figure 4, where green indicates a covert channel easy to mitigate and red indicates a covert channel that is hard to mitigate.

5.1 Unused Registers

Embedded sensors usually contain unused registers that are either reserved or not required by the sensor's current mode of operation. Similar to network-based covert channels, these registers can be exploited for

transferring data.

Reserved Registers: are registers that are neither used by the sensor to publish information (e.g. status flags), nor to store data such as configuration parameters for the sensor. Sensors, such as the HTS221 (STMicroelectronics, 2016) humidity and temperature sensor or the LSM9DS1 (STMicroelectronics, 2015) magnetometer, accelerometer, and gyroscope, contain many such reserved registers. The reserved registers are listed in the respective data-sheets. Although the data-sheets often state that these registers *must not* be changed, they are often still read- *and* writeable.

Unused Registers: are registers that might be used in some sensor operation modes, but are unused in other modes. For instance, many sensors such as the LSM9DS1 sensor allow to set thresholds (e.g. INT_THS_* registers) that are used to activate flags that indicate if the threshold is exceeded. However, if the threshold monitoring is disabled (in the ACT_THS register), the threshold registers are unused, and thus can be used for data transfer in a covert channel. Since the register that indicates which thresholds are monitored is readable, a malicious process easily can determine which threshold registers are unused.

5.1.1 Covert Channel Design

Unused registers facilitate a very simple covert channel design. Information can be written into a register by one process, while the other process reads the register and subsequently confirms reception of data by modifying the same register. In our design, we use the register's MSB as a flag to signal successful reception by the receiver. Therefore, not all bits of a register can be used for data transfer. Such a covert channel that is hiding information in registers is comparable to network-based covert channels that use reserved protocol fields or bits (Rezaei et al., 2013) for hiding transferred data. Depending on the sensor's register size, the amount of data that can be transferred in one transmission varies. Both HTS221 and LSM9DS1 contain 8 bit registers which allows 7 bits of data to be sent in each transmission.

5.1.2 Detectability

Reserved registers can be read and written to; however, writing values to these registers might have an impact on a sensor's correct functionality. Depending on the resulting impact of writing arbitrary values to such a register, covert channels that are based on writing and reading these registers could easily be detected. In comparison to that, a covert channel that is based on not required registers is harder to detect. As long as the register's value has no impact on the sensor's functionality, the covert channel does not influence the sensor's behavior.

5.1.3 Countermeasures

To mitigate a covert channel based on unused registers, various countermeasures can be used. (i) If write access to reserved registers is disabled, these registers cannot be exploited to build covert channels. (ii) Write access to unused registers must be disabled whenever a register is not required in the sensor's current mode of operation. Whenever the register content is required, write access for the respective register can be granted again by the sensor. (iii) Write-only configuration registers could mitigate such covert channels since the receiver would not be capable of reading transmitted data anymore. However, write-only registers also complicate updating the register's value, if the current value would be required first. For example, updating only a certain part of the register such as a threshold's exponent without modifying the remaining bits, requires bit-wise operations, such as AND, OR, and XOR, for registers.

5.2 Configuration Bits

Embedded sensors are configured using so-called configuration registers. In these registers, various different settings are often combined for efficiency reasons. Similarly to exploiting whole registers that are unused, certain bits of these configuration registers can often also be exploited.

Reserved Bits: of configuration registers can be used to transfer data in a covert channel. For example, bits [7:6] in the HTS221 sensor's AV_CONF register are reserved bits that do not influence any configuration state. However, similar to unused registers, the data-sheet states that these bits *must not* be changed to not cause unwanted sensor behavior.

LSBs of Configured Values: such as thresholds can be used to hide transferred data in a covered channel. This approach is similar to hiding data in the LSBs of header fields in network-based covert-channels. If chosen correctly, manipulating the LSBs of, e.g., a threshold value only has a negligible impact on the sensor's expected functionality.

Unused Configuration Bits: can be present in configuration registers if the number of available options

Table 1: OPT3001 ambient light sensor modes of operation.

Mode	Mode[1]	Mode[0]
shutdown	0	0
single-shot	0	1
continuous	1	0
continuous	1	1

is smaller than the maximum number that can be represented by the respective part of the configuration register. For instance, in the OPT3001 (Instruments, 2014) ambient light sensor's configuration register, 2 bits are reserved for a configuration parameter that has three available options. As shown in Table 1, if the MSB (Mode[1]) is set to '1', the LSB (Mode[0]) can be used to transfer data in a covert channel.

5.2.1 Covert Channel Design

A covert channel that is based on exploiting configuration bits can be based on the same principles as a covert channel that exploits unused registers. In a first step, the targeted configuration bits need to be determined. After that, these bits are used to transfer data in a covert channel by encoding data in these available bits. The recipient of data reads the respective bits and confirms if data is successfully read. Similarly to exploiting unused registers, one bit is required that is used as status flag for confirming that the receiver successfully read the transferred data. Therefore, such a covert channel requires at least *two* available bits. If only one bit is available, both sender and receiver must be synchronized by other measures such as a clock which slows down the covert channel. Compared to a covert channel that is able to exploit a whole unused register, a covert channel that is only able to utilize some bits of a register will provide lower covert channel data rates.

5.2.2 Detectability

Depending on the configuration bits that are exploited to build the covert channel, the detectability also varies. If reserved bits are used, correct sensor functionality might be influenced, and this may lead to easy detection of the covert channel. If bits that only have a minimal or no impact on the sensor's functionality are manipulated, the covert channel is harder to detect. However, toggling configuration parameters might cause sensors to restart their current measurement. Therefore, data transfer must be timed in order to minimize such detectable effects.

5.2.3 Countermeasures

To mitigate covert channels that are based on exploiting configuration bits, the following countermeasures can be implemented on a sensor. (i) Disabling write access to reserved bits mitigates misuse of these bits by covert channels. (ii) Write-only configuration registers mitigate covert channels that are based on exploiting configuration bits since the recipient of data is unable to read the register. However, similar to countermeasures for unused registers (Subsection 5.1.3), bit-wise operations, such as AND, OR, and XOR, will be required for registers.

5.3 Triggering Sensors

Both register exploit methods (unused registers and configuration bits) require read- and write-access to the same register. However, as briefly discussed in the respective countermeasure subsections, countermeasures to mitigate these exploits can easily be implemented in software or hardware. Nevertheless, exploiting embedded sensors via registers is still possible even if these countermeasures are implemented on a sensor. If there are *read-only* registers at a sensor that can be updated by certain *events*, and these events can be triggered by one process, a covert channel according to the definition shown in Figure 1 can still be built. For example, on most sensors status flags in registers are used to indicate a finished sensing process. If the sensor is not operated in a continuous sensing mode but in a single-shot mode, one process is capable of updating these status flags by triggering sensor readings. The status flags can then be used to encode information in various ways. For example, information can be encoded in timing differences, or, if multiple status bits exist, directly in these status bits.

Timing Differences: between sensor readings can be used to encode information. For example, a binary '1' could be transferred by requesting sensor readings with a time interval between readings of 100 ms (10 Hz). A binary '0' would then be transferred by using a different timing interval, for example, 50 ms (20 Hz). The receiver then needs to observe the status bit to get timing intervals and to infer the corresponding data. However, the drawback of such an approach is that sender and receiver need to be synchronized to guarantee precise timings. In addition, the receiver would need to poll status bits with a high frequency.

Table 2: Status flags for two sensors and the available states for a 2-bit word and a 1-bit word respectively.

S1	S2	2-bit word	1-bit word
0	0	'00' *and* no data	no data
0	1	'01'	'0'
1	0	'10'	'1'
1	1	'11' *and* sensor ready	sensor ready

Directly Encoding Information in Status Bits: is possible if there is more than one status bit that can be triggered by the sender. For example, sensors that are capable of sensing more than one physical property also contain multiple status flags to indicate a finished sensing process for each property. For example, the `HTS221` (STMicroelectronics, 2016) temperature and humidity sensor includes status registers for both physical properties. Using both registers, a 2-bit word can be encoded by triggering either no, one of both, or both sensors simultaneously. However, similarly to measuring timing differences, this approach would require precise synchronization between sender and receiver to distinguish a transferred '00' from the status flags default value that often is also set to '00'. To discard this requirement, data can be encoded by triggering one sensor to transmit a binary '1', and by triggering the other sensor to transmit a binary '0'. Both mentioned approaches are compared in Table 2.

5.3.1 Covert Channel Design

Since covert channels based on encoding information in status bits do not require synchronization between processes, we consider this type of covert channel more practical. Therefore, we are going to discuss a covert channel based on transmitting a 1-bit word using the status flags of two distinct sensors. In its default setting, the sensor's status flag is set to '0' and indicates that no sensor reading is ready at the moment. If a sensor reading is available, the respective status flag is set to '1'. If the sensor's measured value is read, the status flag is reset to '0' again. In our covert channel, the sender triggers both sensors. After the sensing process is completed, the sender reads one of the two sensor measurements to reset the respective status flag. Information is encoded as a 1-bit word according to Table 2. The receiver can observe the same status flags, and thus receive the transmitted information. The reception of data is confirmed by the receiver by resetting both status flags. Using the encoding shown in Table 2 in an example, the binary sequence '11010' would be encoded by resetting the status flags from the sensors `S1`, `S1`, `S2`, `S1`, `S2` respectively. Since the sensing process requires a certain amount of time, this covert channel's data rate is lower compared to covert channels that directly write information into a sensor's registers.

5.3.2 Detectability

Compared to directly writing into a sensor's registers, a covert channel that is based on triggering sensors is harder to detect. Since only the sender triggers sensor readings, while the receiver is only observing status flags, no malicious activity might be noticed when monitoring sensor activities (Mirzamohammadi et al., 2017). The behavior that can be observed in such a case are two processes where one process is using sensors frequently, while the other process is checking the availability of these sensors. If the roles of sender and receiver are switched (Section 7.2), the covert channel's behavior is comparable to two processes that alternately access the same sensors.

5.3.3 Countermeasures

To mitigate covert channels that are based on triggering sensors, more complex countermeasures are required in comparison to covert channels that exploit read- and write-able registers. In principle, any link between the event that can be triggered by a process and observable information needs to be removed. To mitigate all three discussed covert channels, a sensor management instance that encapsulates sensor access is required. As an example, the Android Sensor Manager (Milette and Stroud, 2012) only allows processes to register for sensor data they are interested in. The manager then determines the superset of all requested sensor configurations. Whenever a new sensor reading is available, all registered processes are notified via an interrupt. Therefore, such a managed approach would remove any status flag that indicates available sensor readings or exceeded thresholds, and thus mitigates covert channels based on such information.

6 MANAGED SENSOR EXPLOITS

Contrary to the previous sensor-based covert channel designs, Android uses a managed sensor approach (Milette and Stroud, 2012) where processes need to subscribe to a sensor manager to get sensor readings based on events. Thus, access to sensor registers as well as manually triggering sensor readings is not possible in Android. However, in this section we demonstrate two different approaches how we exploit the Android sensor manager to build sensor-based covert channels based on triggering sensors.

When registering a listener for any sensor that is supported by the respective hardware platform us-

ing Android's built in sensor manager, the method `SensorManager.registerListener()` is used that takes the type of sensor as well as a *sampling period* as parameters. As stated in Android's API documentation, this sampling period is only a *suggested delay* that might be altered by *other applications*. Typically, the sampling period can be set based on values predefined in Android (normal, UI, game, fastest) or specified arbitrarily. Thus, if one process registers a listener with a given sampling period which is then influenced by another process, data can be transferred between these processes using this covert channel.

6.1 Covert Channel Design

Based on the observation that a process may influence the sampling period of other processes, we present two methods for covert channels in Android.

Frequency Encoded. Android's sensor manager provides sensor measurements to all registered processes with the lowest sampling period specified by all registered processes. If the receiver registers a listener with a sampling period that is even lower than the current lowest sampling frequency, it signals to the sender that it now is ready to receive data. That is, by specifying the sensors's lowest sampling period, all processes now receive sensor measurements with the sampling period specified by the receiver. The sender now can encode information by registering with either the same sampling period as the receiver or with an ever lower sampling period, as shown in Figure 5. Thus, information is encoded by different sampling periods or sensing frequencies.

Outlier Intervals. Whenever a new listener is registered using Android's sensor manager, one sensor measurement is provided with a time interval to the previous measurement that can clearly be detected as outlier. Depending on the hardware, we observed either outliers of too low (Figure 5) or too high (Figure 6) sampling periods. That is, even if a process already is registered for a sensor using the lowest possible sampling period supported by the hardware, information can still be transferred by registering new listeners to provoke such outliers. As shown in Figure 6, the interval between outliers can then be used to encode information that is transferred over the respective covert channel.

6.2 Detectability

Covert channels based on registering sensor listeners cannot easily be detected since the switching of

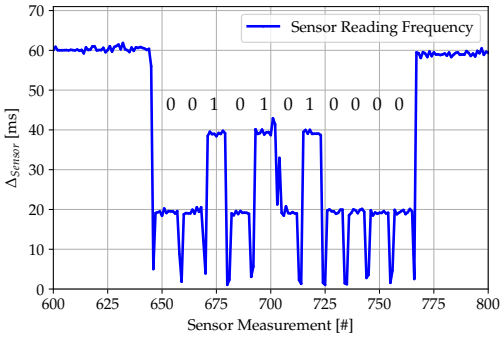


Figure 5: Different sensing intervals in Android.

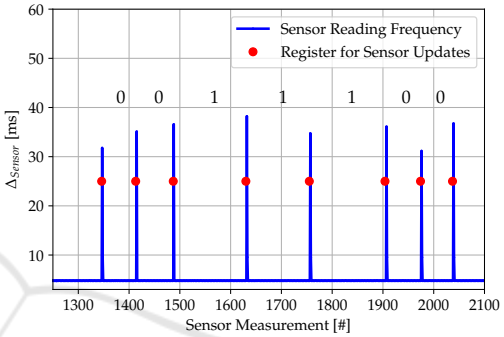
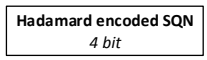


Figure 6: Registering sensor listeners in Android.

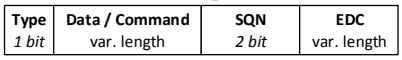
sampling periods due to other processes registering and un-registering listeners is expected behavior. A process that is minimized is expected to un-register its listener, while re-registering them if the process is activated again. However, if sensor access is audited (Han et al., 2017), malicious access patterns could be detected if the auditing tool is trained accordingly. In contrast, popular code analysis tools such as FlowDroid (Arzt et al., 2014) are currently not capable of detecting our presented covert channels. However, also these tools can be appropriately trained such that the presented covert channels can be found.

6.3 Countermeasures

To mitigate the presented covert channels, changes to Android's sensor manager need to be implemented. If arbitrary sampling periods are banned and only predefined sampling periods are used, the sampling periods need to be defined such that they are multiples of each other. For example, if the predefined sampling period *fastest* is defined as 10 ms, *game* could be defined as 20 ms, *UI* as 80 ms, and *normal* as 160 ms. By doing so, the sensor internally can provide sensor measurements with the system's lowest specified sampling period, while each process receives sensor measurements with its specified sampling period only.



(a) REQ packet.



(b) RES packet.

Figure 7: Packet structures of REQ and RES packets.

7 TEST FRAMEWORK

To facilitate easier vulnerability testing of embedded sensors, we present a modular covert channel framework that is structured into the following four abstraction layers. Hardware specific aspects are implemented in a *hardware abstraction* layer that is comprised of the following three sub-layers: (i) The lowest layer (*access abstraction*) implements access to embedded sensors through various technologies, such as I2C or SPI. (ii) The *sensor abstraction* layer implements sensor specific aspects such as register mappings. (iii) All sensor-based exploits that are presented in Section 5 are implemented in the *exploit abstraction* layer. (iv) Protocol specific functionality (Subsections 7.1 – 7.3) is implemented in the *covert channel abstraction* layer.

7.1 Error Detection and Correction

As other processes might also access sensors, a covert channel needs to be considered a *noisy channel*. However, an error-free data transmission through a noisy channel can be achieved if the data is sufficiently encoded by appropriate coding schemes (Shannon, 1948). In our covert channel framework, we use Error Correcting Codes (ECCs) as well as Error Detecting Codes (EDCs) to reliably transfer our messages, as will be discussed in Section 7.2.

7.2 Packet Structure and Flow

The packet structures of request (REQ) and response (RES) packets in our approach are shown in Figure 7.

REQ Packet. The only information contained in a REQ packet is the Sequence Number (SQN) encoded by a Hadamard ECC. In general, a Hadamard ECC encodes a k bit message in a 2^k bit codeword. Due to the exponential relationship between payload size and codeword size, we use a 2 bit SQN, resulting in a total size of 4 bits. The Hadamard ECC is proven optimal for $k \leq 7$ (Bouyukliev and Jaffe, 2001). In case of transmission errors, the ECC ensures that information can be recovered and errors are detected as long as *less than half* of the bits are flipped.

Table 3: Supported commands.

Code	Description
0	Increment packet data size (Section 7.3)
1	Decrement packet data size (Section 7.3)
2	Stop data transmission
3	Reverse data direction

Table 4: Valid size options.

Data [bits]	EDC [bits]	Packet [bits]
5	3	11
13	4	20
29	5	37
61	6	70

RES Packet. All RES packets start with a type field that specifies whether the message contains *data* or a *command*. Commands that are supported by our covert channel framework are shown in Table 3. Identical to REQ packets, RES packets also contain a 2 bit SQN. Due to their length, we do not use ECCs for RES packets. Instead, a Berger EDC (Berger, 1961) is used to detect transmission errors that need to be handled by the communication principle implemented in our framework. A k bit Berger code is capable of checking a maximum of $n = 2^k - 1$ bits information. Thus, the resulting data/command and EDC lengths can be derived from the packet's total size (Table 4).

To manage communication flow in our presented covert channels, we employ a request/response mechanism. A successful request/response cycle is comprised of the reception of a REQ packet by the sender and the reception of the RES packet by the receiver. Similar to HTTP, the actual data that is transferred is contained in the RES packet. Both, REQ and RES packets contain a SQN that is used to manage communication flow. Matching REQ and RES packets can be identified by their matching SQN. The receiver increases the SQN after successfully receiving the corresponding RES packet, as shown in Figure 8. By repeatedly sending the same SQN, receiver and sender can indicate that the expected packet was not successfully received. Retransmissions are caused in three scenarios:

1. When establishing the covert channel, the sender might not be ready to send the requested data and no matching RES packet is sent to the receiver's initial REQ packet. The receiver continuously transmits this initial packet until a covert channel is established, thereby synchronizing the states of receiver and sender.
2. A REQ packet can be lost due to a noisy data

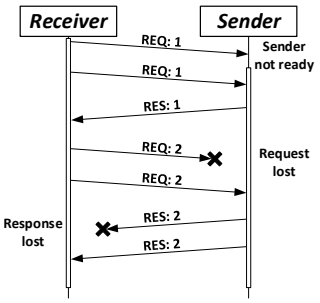


Figure 8: Data flow and handling of lost packets.

channel. The receiver repeatedly sends its REQ packet until a matching RES packet is received.

3. A RES packet can also be lost due to a noisy data channel. The sender repeatedly sends its RES packet until a REQ packet with an incremented SQN is received.

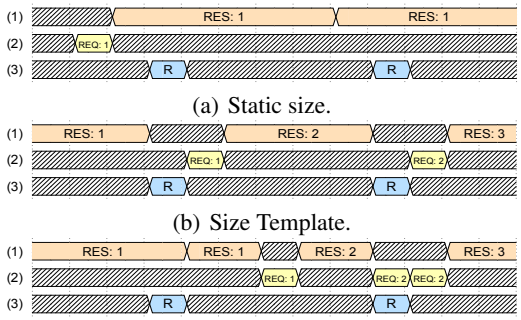
The commands supported by our covert channel framework (Table 3) also include two commands related to data flow. (i) To indicate the end of an ongoing data transfer, the sender sends a stop command to the receiver. (ii) The roles of sender and receiver can be reversed by sending the respective command. Thus, our covert channel also supports *bidirectional* communication.

7.3 Adaptive Packet Length

Information is transferred by interacting with a sensor in all presented sensor-based covert channels. Thus, other processes that also interact with the *same* sensor might introduce bit errors into our covert channel. These bit errors can be detected by our approach due to RES packets containing an EDC. However, as shown in Figure 9(a), this leads to frequent retransmissions of large chunks of information.

As already briefly discussed in Section 7.2, our covert channel supports dynamic packet sizes that can be used to minimize negative effects caused by bit errors. In addition, our framework also supports finding packet size templates for finding optimal sizes.

Finding Size Template. Although decreasing the size of RES packets may lead to less retransmissions, the overhead increases due to additional REQ packets (Figure 9(c)). Therefore, we propose analysing the potential covert channel before starting any data transmission. That is, the sender only observes the channel for sensing activity and tries to calculate an optimal packet size if other processes are accessing the sensor frequently (Figure 9(b)).



(a) Static size.

(b) Size Template.

(c) Dynamic Size.

Figure 9: Different transfer modes. (1) sender, (2) receiver, and (3) another process reading the sensor (R).

Dynamic Packet Size. If an optimal packet size cannot be determined, e.g., if another process is accessing a sensor infrequently, a covert channel might be unable to transfer any data. Therefore, we introduce a dynamic packet size approach. Whenever bit errors are detected using the EDC, it is assumed that another process is accessing the same sensor as the current covert channel. As a consequence, the packet size is reduced, which is indicated by sending the respective command (Table 3). Reducing the packet size results in a lower amount of retransmitted data as shown in Figure 9(c). Packet sizes are increased again if a certain amount of successfully transferred RES packets is exceeded.

8 EVALUATION

To evaluate the presented covert channels and the functionality of our covert channel framework, we use the following three hardware platforms and OSs:

1. *CC2650 SensorTag*; *TI-RTOS* 2.20
2. *Raspberry Pi 3, Sense HAT*; *Raspbian Strech*
3. *OnePlus 5* & *Android Emulator*; Android 8.0

8.1 Covert Channel Data Rates

To evaluate data rates and to validate the classification shown in Figure 4, we measured data rates on different platforms. The data rates we achieved in our evaluation are shown in Figure 10. As can be seen there, only the Raspberry Pi 3 with attached Sense HAT allows all three covert channels to be implemented. On the CC2650 SensorTag, only unused registers and configuration bits can be exploited. Contrary to that, the only side-channel that can be exploited in Android is triggering sensors. Both covert channels implemented on the CC2650 SensorTag running TI-RTOS

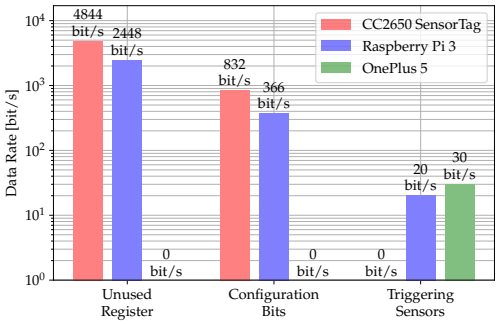


Figure 10: Evaluation of covert channel data rates.

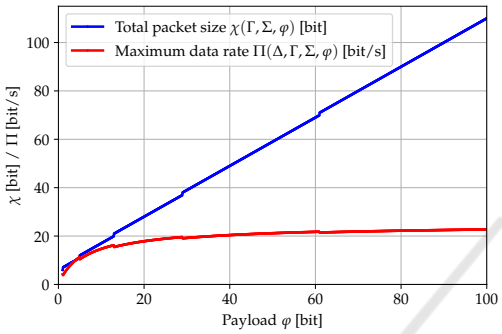


Figure 11: Covert channel data rate according to (1).

achieve higher data rates compared to our implementations on the Raspberry Pi 3 due to the deterministic scheduling of TI-RTOS. Processes in TI-RTOS are scheduled alternately, compared to the indeterministic scheduling of Raspbian. Compared to that, the covert channel based on triggering sensors does not offer a very high data rate. However, the maximal achievable covert channel data rate Π depends on the time Δ required for a sensor reading. Π is then defined in dependence of the payload size φ, and the total round trip size $\chi(\varphi,\Gamma)$ that is comprised of all protocol fields (Γ), EDC ($\lceil \log_2(\Gamma+\varphi) \rceil$), and ECC ($\Sigma$).

$$\Pi(\Delta,\Gamma,\Sigma,\varphi) = \varphi/(\chi(\Gamma,\Sigma,\varphi)\Delta) \quad (1)$$

$$\chi(\Gamma,\Sigma,\varphi) = \Gamma+\Sigma+\varphi+\lceil \log_2(\Gamma+\varphi) \rceil \quad (2)$$

These two functions are evaluated and plotted for a range of different payload sizes in Figure 11. The covert channel data rate converges to a value of 24 bit/s on a Raspberry Pi 3 platform. Since we do not consider any delay caused by the bus or program execution in (1), we consider our achieved covert channel data rate of 20 bit/s on that platform, close to the theoretical maximum.

8.2 EDC Retransmission Functionality

To evaluate the implemented EDC and retransmission functionality (Section 7.1), as well as the overhead

(a) Source. (b) Received. (c) Errors.

Figure 12: EDC and retransmissions disabled.

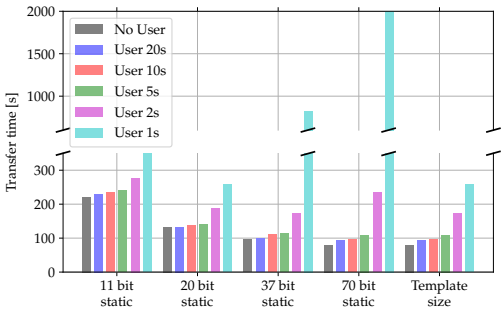


Figure 13: Evaluation of different packet sizes.

resulting from such retransmissions we simulated a noisy covert channel. As evaluation setting, a covert channel based on triggering sensors on a Raspberry Pi Sense HAT was used. Noise on these registers was introduced by running a process that accesses the same sensor as our covert channel every 10 s. We then transferred an image (4 kB) over this noisy channel. Figure 12 shows the original image as well as the received image that contains roughly 100 pixel errors if our EDC based retransmission of packets is disabled. Enabling these features results in an error-free image being transferred over our covert channel.

Overhead. Enabling EDC and retransmissions in a setting *without* noise introduces an transmission time overhead of roughly 6%. While the image with disabled EDC and retransmissions is transferred in 2998 s, transferring the image with EDC and retransmissions requires 3169 s. If noise is introduced by a process that is accessing the sensor every 10 s, transfer time increases to 3989 s in this setting as the noise requires 123 of 1062 packets to be retransmitted.

8.3 Static / Dynamic Packet Size

To evaluate the performance of different packet sizes that are supported by our framework (Table 4), we test these packet sizes with noise generated by different *noise profiles*. We test profiles with *no* interfering user actions as well as well as profiles where users access the same sensor that our covert channel is using. The evaluation results in Figure 13 highlight that there is no packet size that is capable of providing the fastest transfer time for each noise profile. However, if sensor access by the user is cyclic, our template method is able to provide best results for each noise profile.

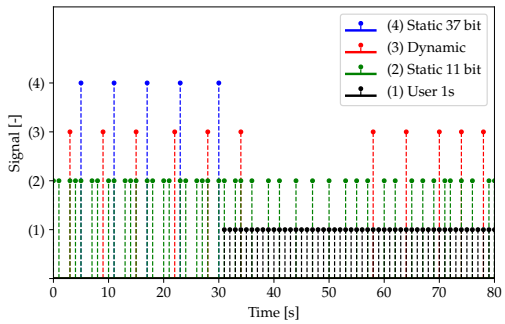


Figure 14: Evaluation of dynamic packet size.

If noise is introduced by a user that is accessing the sensor infrequently, or if no user was present during our template phase, the dynamic switching of packet sizes implemented in our covert channel framework is capable of ensuring a reliable and fast data transfer. Figure 14 evaluates such a scenario where a user starts to access the same sensor that our covert channel is using after the data transfer already started. In our evaluation, we compare two static packet sizes of 11 bits and 37 bits as well as our dynamic packet size mechanism that is configured to switch between packet sizes of 11 bits, 20 bits and 37 bits. We can identify the following three phases shown in Figure 14, where each data point represents the successful submission of one data packet or an interfering sensor access, respectively. (i) In the interval from 0 s to 30 s no interfering sensor access occurs. The dynamic packet size is set to 37 bits since this ensures the fastest data transfer. Both static variants transfer packets without errors. (ii) After 30 s, the user process starts to interfere, and thus packets with a size of 37 bits cannot be transferred any more due to the generated noise, as shown in Figure 9(a). Smaller packets are successfully transferred with a lower probability in this phase. Our dynamic packet size approach is now sending command messages to decrease the packet size. (iii) Starting at 58 s, the dynamic packet size approach can successfully transfer packets again due to decreasing the packet size.

8.4 Comparison to State-of-the-art

The currently fastest data rates were reported for cache- and DRAM-based covert channels. The Flush+Flush (Gruss et al., 2016) cache-based covert channel is capable of data rates of up to 3.8 Mbit/s while the fastest DRAM-based covert channel achieves data rates of up to 2 Mbit/s. However, when comparing these covert channel data rates, it has to be considered that modern CPU caches and DRAMs are capable of achieving bandwidths in the range of 20 Gbit/s to 100 Gbit/s (Molka et al., 2015). That is, the presented covert channels use roughly 1% of the technology's possible bandwidth. Other memory based covert channels (Luo et al., 2015) provide a data rate of 747 bit/s and are slower than our fastest covert channel. The fastest reported network-based covert channel supports data rates of up to some kbit/s (Zander et al., 2007b); however, it also has to be considered for these covert channels that the network technology would provide a bandwidth of at least 100 Mbit/s. Other sensor-based covert channels reported data rates of 345 bit/s, which matches our second fastest covert channel. If considering the relatively low bandwidth provided by the I2C bus that is used in our evaluation, we claim that our presented covert channel implementations provide highly competitive data rates and bus utilization compared to the state-of-the-art.

9 CONCLUSION

In this paper, we presented novel exploits that target unsecured sensor interfaces. We use these exploits to demonstrate three different sensor-based covert channels that provide a trade-off between the achievable covert channel data rate and the likeliness of detecting the malicious behavior. Our fastest covert channel provides data rates of up to 4844 bit/s, while the slowest covert channel only provides a data rate of 20 bit/s but will not be distinguishable from normal user behavior. Our presented Android covert channels are not detected by state-of-the-art code analysis tools. We do not claim that the presented list of exploits is exhaustive, but rather believe that other issues can and will be found in current embedded sensors. To facilitate testing other platforms for security weaknesses, we provide our covert channel framework on GitHub. All countermeasures suggested in this paper can easily be implemented on embedded sensors. Therefore, this paper highlights the importance of implementing such countermeasures to mitigate sensor-based covert channels and to prevent future sensor-related security issues.

ACKNOWLEDGEMENTS

This work has been performed within the IoSense (http://iosense.eu) project. This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 692480. This Joint Undertaking receives support from the European Union's Horizon

2020 research and innovation programme and Germany, Netherlands, Spain, Austria, Belgium, Slovakia. IoSense is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2016 and April 2019. More information: https://iktderzukunft.at/en/.

REFERENCES

Ahsan, K. and Kundur, D. (2002). Practical Data Hiding in TCP/IP. In *Proceedings of the Workshop on Multimedia Security at ACM Multimedia*, pages 1–8. ACM.

Ameri, A. and Johnson, D. (2017). Covert Channel over Network Time Protocol. In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, pages 62–65. ACM.

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., and McDaniel, P. (2014). FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *ACM SIGPLAN Notices - PLDI '14*, 49(6):259–269.

Aviv, A. J., Sapp, B., Blaze, M., and Smith, J. M. (2012). Practicality of Accelerometer Side Channels on Smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 41–50. ACM.

Berger, J. M. (1961). A Note on Error Detection Codes for Asymmetric Channels. *Information and Control*, 4(1):68–73.

Bouyukliev, I. and Jaffe, D. B. (2001). Optimal binary linear codes of dimension at most seven. *Discrete Mathematics*, 226(1-3):51–70.

Brouchier, J., Kean, T., Marsh, C., and Naccache, D. (2009). Temperature Attacks. *IEEE Security & Privacy*, 7(2):79–82.

Cabuk, S., Brodley, C. E., and Shields, C. (2004). IP Covert Timing Channels: Design and Detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 178–187. ACM.

Carrara, B. and Adams, C. (2016). Out-of-Band Covert Channels—A Survey. *ACM Computing Surveys (CSUR)*, 49(2):23.

Chi, Q., Yan, H., Zhang, C., Pang, Z., and Da Xu, L. (2014). A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment. *IEEE transactions on industrial informatics*, 10(2):1417–1425.

Derler, P., Lee, E. A., and Vincentelli, A. S. (2012). Modeling Cyber–Physical Systems. *Proceedings of the IEEE*, 100(1):13–28.

Frikha, L., Trabelsi, Z., and El-Hajj, W. (2008). Implementation of a Covert Channel in the 802.11 Header. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC'08. International*, pages 594–599. IEEE.

Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., and Yarom, Y. (2016). ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1626–1638. ACM.

Giffin, J., Greenstadt, R., Litwack, P., and Tibbetts, R. (2002). Covert Messaging through TCP Timestamps. In *International Workshop on Privacy Enhancing Technologies*, pages 194–208. Springer.

Gruss, D., Maurice, C., Wagner, K., and Mangard, S. (2016). Flush+Flush: A Fast and Stealthy Cache Attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer.

Guri, M., Monitz, M., Mirski, Y., and Elovici, Y. (2015). BitWhisper: Covert Signaling Channel between Air-Gapped Computers using Thermal Manipulations. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 276–289. IEEE.

Han, W., Cao, C., Chen, H., Li, D., Fang, Z., Xu, W., and Wang, X. S. (2017). senDroid: Auditing Sensor Access in Android System-wide. *IEEE Transactions on Dependable and Secure Computing*, (1):1–1.

Instruments, T. (2014). OPT3001 Ambient Light Sensor (ALS). [Online; accessed 10-March-2018].

Ji, L., Jiang, W., Dai, B., and Niu, X. (2009). A Novel Covert Channel Based on Length of Messages. In *Information Engineering and Electronic Commerce, 2009. IEEC'09. International Symposium on*, pages 551–554. IEEE.

Kim, C. H. and Quisquater, J.-J. (2007). How can we overcome both side channel analysis and fault attacks on RSA-CRT? In *Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on*, pages 21–29. IEEE.

Kwon, C., Liu, W., and Hwang, I. (2013). Security Analysis for Cyber-Physical Systems against Stealthy Deception Attacks. In *American Control Conference (ACC), 2013*, pages 3344–3349. IEEE.

Lampson, B. W. (1973). A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615.

Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. (2015). Last-Level Cache Side-Channel Attacks are Practical. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 605–622. IEEE.

Longo, J., De Mulder, E., Page, D., and Tunstall, M. (2015). SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip. In *Intl. W. on CHES*, pages 620–640. Springer.

Luo, C., Fei, Y., Luo, P., Mukherjee, S., and Kaeli, D. (2015). Side-Channel Power Analysis of a GPU AES Implementation. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 281–288. IEEE.

Maurice, C., Weber, M., Schwarz, M., Giner, L., Gruss, D., Alberto Boano, C., Mangard, S., and Römer, K. (2017). Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium*, NDSS. The Internet Society.

Milette, G. and Stroud, A. (2012). *Professional Android Sensor Programming*. John Wiley & Sons.

Mirzamohammadi, S., Chen, J. A., Sani, A. A., Mehrotra, S., and Tsudik, G. (2017). Ditio: Trustworthy Auditing of Sensor Activities in Mobile & IoT Devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, page 14. ACM.

Molka, D., Hackenberg, D., Schöne, R., and Nagel, W. E. (2015). Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 739–748. IEEE.

Nguyen, A., Alqurashi, R., Raghebi, Z., Banaei-Kashani, F., Halbower, A. C., and Vu, T. (2016). A Lightweight and Inexpensive In-ear Sensing System For Automatic Whole-night Sleep Stage Monitoring. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 230–244. ACM.

Osvik, D. A., Shamir, A., and Tromer, E. (2006). Cache Attacks and Countermeasures: The Case of AES. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454.

Perrig, A., Stankovic, J., and Wagner, D. (2004). Security in Wireless Sensor Networks. *Communications of the ACM*, 47(6):53–57.

Pessl, P., Gruss, D., Maurice, C., Schwarz, M., and Mangard, S. (2016). DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security Symposium*, pages 565–581.

Poeplau, S., Fratantonio, Y., Bianchi, A., Kruegel, C., and Vigna, G. (2014). Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*, NDSS, pages 23–26. The Internet Society.

Rezaei, F., Hempel, M., Peng, D., Qian, Y., and Sharif, H. (2013). Analysis and Evaluation of Covert Channels over LTE Advanced. In *WCNC Intl. Conf., 2013 IEEE*, pages 1903–1908. IEEE.

Sadeghi, A.-R., Wachsmann, C., and Waidner, M. (2015). Security and Privacy Challenges in Industrial Internet of Things. In *Proceedings of the 52Nd Annual Design Automation Conference*, pages 54:1–54:6. ACM.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423.

Shrivastava, A., Jain, P., Demetriou, S., Cox, P., and Kim, K.-H. (2017). CamForensics: Understanding Visual Privacy Leaks in the Wild. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, page 13. ACM.

Spreitzer, R. (2014). PIN Skimming: Exploiting the Ambient-Light Sensor in Mobile Devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 51–62. ACM.

Srbinovska, M., Gavrovski, C., Dimcev, V., Krkoleva, A., and Borozan, V. (2015). Environmental parameters monitoring in precision agriculture using wireless sensor networks. *Journal of cleaner production*, 88.

STMicroelectronics (2015). LSM9DS1 iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer. [Online; accessed 10-March-2018].

STMicroelectronics (2016). HTS221: Capacitive digital sensor for relative humidity and temperature. [Online; accessed 10-March-2018].

Suo, H., Wan, J., Zou, C., and Liu, J. (2012). Security in the Internet of Things: A Review. In *Computer Science and Electronics Engineering (ICCSEE), 2012 Intl. Conf. on*, volume 3, pages 648–651. IEEE.

Tuptuk, N. and Hailes, S. (2015). Covert Channel Attacks in Pervasive Computing. In *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on*, pages 236–242. IEEE.

Wang, Z. and Lee, R. B. (2006). Covert and Side Channels Due to Processor Architecture. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 473–482. IEEE.

Wu, J., Ding, L., Wang, Y., and Han, W. (2011). Identification and Evaluation of Sharing Memory Covert Timing Channel in Xen Virtual Machines. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 283–291. IEEE.

Xiao, J., Xu, Z., Huang, H., and Wang, H. (2013). Security Implications of Memory Deduplication in a Virtualized Environment. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE.

Yarom, Y. and Falkner, K. (2014). Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Sec. Symp.*, pages 719–732.

Yi, X., Bouguettaya, A., Georgakopoulos, D., Song, A., and Willemson, J. (2016). Privacy Protection for Wireless Medical Sensor Data. *IEEE Transactions on Dependable and Secure Computing*, 13(3):369–380.

Yu, J., Zhao, J., Chen, Y., and Yang, J. (2015). Sensing Ambient Light for User Experience-Oriented Color Scheme Adaptation on Smartphone Displays. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 309–321. ACM.

Zander, S., Armitage, G., and Branch, P. (2007a). A Survey of Covert Channels and Countermeasures in Computer Network Protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57.

Zander, S., Armitage, G., and Branch, P. (2007b). An Empirical Evaluation of IP Time To Live Covert Channels. In *Networks, 2007. ICON 2007. 15th IEEE International Conference on*, pages 42–47. IEEE.

Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. (2012). Cross-VM Side Channels and Their Use to Extract Private Keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 305–316. ACM.