TrustAV: Practical and Privacy-Preserving Malware Analysis in the Cloud

Dimitris Deyannis*
FORTH-ICS
Heraklion, Greece
deyannis@ics.forth.gr

Eva Papadogiannaki*
FORTH-ICS
Heraklion, Greece
epapado@ics.forth.gr

Giorgos Kalivianakis*
FORTH-ICS
Heraklion, Greece
gkaliv@ics.forth.gr

Giorgos Vasiliadis
FORTH-ICS
Heraklion, Greece
gvasil@ics.forth.gr

Sotiris Ioannidis
FORTH-ICS
Heraklion, Greece
sotiris@ics.forth.gr

ABSTRACT

While the number of connected devices is constantly growing, we observe an increased incident rate of cyber attacks that target user data. Typically, personal devices contain the most sensitive information regarding their users, so there is no doubt that they can be a very valuable target for adversaries. Typical defense solution to safeguard user devices and data, are based in malware analysis mechanisms. To amortize the processing and maintenance overheads, the outsourcing of network inspection mechanisms to the cloud has become very popular recently. However, the majority of such cloud-based applications usually offers limited privacy preserving guarantees for data processing in third-party environments.

In this work, we propose TrustAV, a practical cloud-based malware detection solution destined for a plethora of device types. TrustAV is able to offload the processing of malware analysis to a remote server, where it is executed entirely inside, hardware supported, secure enclaves. By doing so, TrustAV is capable to shield the transfer and processing of user data even in untrusted environments with tolerable performance overheads, ensuring that private user data are never exposed to malicious entities or honest-but-curious providers. TrustAV also utilizes various techniques in order to overcome performance overheads, introduced by the Intel SGX technology, and reduce the required enclave memory –a limiting factor for malware analysis executed in secure enclave environments– offering up to *3x* better performance.

CCS CONCEPTS

• **Security and privacy → Systems security**; **Intrusion detection systems**; • **Networks → Cloud computing**.

*Also with the Department of Computer Science, University of Crete, Greece

CODASPY '20, March 16–18, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3375748

KEYWORDS

Intel SGX, trusted execution environment, antivirus, data processing, mobile, privacy

ACM Reference Format:
Dimitris Deyannis, Eva Papadogiannaki, Giorgos Kalivianakis, Giorgos Vasiliadis, and Sotiris Ioannidis. 2020. TrustAV: Practical and Privacy-Preserving Malware Analysis in the Cloud. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3374664.3375748

1 INTRODUCTION

Malware and cyber threats are continually evolving, urging the protection of connected machines [40]. More specifically, Cisco's forecast predicts that the average number of devices and connections per capita will grow from 2.4 in 2017 to 3.6 by 2020, globally [3]. In addition, the percentage of usage of mobile devices grows and it is expected that smartphones will account for 50 percent of total global Internet traffic by 2022, up from 23 percent in 2017 [3]. In the meantime, cyber threats are constantly evolving, bypassing state-of-the-art countermeasures. For instance, Symantec's monthly threat report presents a 12% ransomware activity increase only in a month time period (February to March 2019) [6]. It is obvious that malware defense software has become one of the most popular tools, since users need to mitigate any attack that aims to compromise their personal devices, protecting their transactions and sensitive data. In this work, we propose TrustAV, a flexible and practical malware detection solution. TrustAV offloads the processing of malware analysis to a remote server and it is offered as a cloud-based solution. Recently, the advent of cloud computing has led to the outsourcing of many middlebox applications, including deep packet inspection [12] and virus scanning [23]. The advantages are numerous, such as lower costs spent on equipment, operation and maintenance, better performance and scalability. However, offloading functionality and sensitive personal data (e.g. user network traffic or files) to a possibly untrusted third-party entity, automatically adds an extra layer to the attack surface. To tackle with this possibility and limit the risk, in TrustAV, the remote server is equipped with hardware assisted enclave support (i.e. the Intel SGX technology). Operating in a trusted execution environment (such as Intel SGX) shields both computation and data. The primary aim of our work is to tackle the problem of poor privacy preserving guarantees in state-of-the-art

cloud-based malware analysis solutions. In addition, through our cloud-based implementation, TrustAV can be used on a plethora of devices, either desktop or mobile, using a thin client application. With TrustAV, we make the following contributions:

- We propose a practical cloud-based malware detection solution that provides users with strong privacy preserving guarantees regarding the processing of their personal and sensitive data remotely by utilizing hardware assisted enclaves.
- We identify and present the performance constraints that are introduced by the Intel SGX technology, specifically in relevant signature-based analysis systems (such as intrusion detection or malware detection systems).
- We reduce the enclave memory footprint of our implementation, a limiting parameter for the majority of signature-based solutions in the state-of-the-art, by developing a caching scheme that eliminates EPC swapping and offers up to 3x speedup.
- Our proposed architecture enables the protection of signature-based automata that exceed the enclave memory limits in architectures where swapping of protected memory pages is not supported.

2 BACKGROUND

2.1 Threat model

Today, organizations and enterprises tend to outsource data processing workloads to cloud providers. Providing applications as-a-service has become a very convenient trend due to lower cost and maintenance complexity. Still, these workloads contain important information about the user. More specifically, a malware detection tool has privileged access among the user files, e-mails and network traffic. Processing such sensitive information needs to be taken seriously by complying to security and privacy preserving standards in order to guarantee confidentiality. Yet, how protected can users feel regarding the safety of their data, especially when handled by parties other than themselves? We define three different entities: *(i)* the TrustAV client, *(ii)* the TrustAV server and *(iii)* the cloud provider. The malware scanning engine lives inside the TrustAV cloud-based server, which communicates with the clients through a network connection. We assume that a TrustAV client is installed and initially executed when the device is in a clean state, so no malicious executable has taken the control of the client or the device prior to the execution of our malware detection system. The environment that hosts the TrustAV server is considered untrusted, since there is no control over the operating system, the hypervisor, the drivers, the management stack, the system's memory, I/O devices, etc. Furthermore, even in a fully healthy environment, there is always the possibility of an honest-but-curious cloud provider, willing to learn and extract information regarding the users or the system utilization. In this work, we safeguard both users and TrustAV system from the aforementioned conditions. Obviously, the client and the server are required to safeguard the transmission and the processing of the user's data. For the purpose of completeness, we assume an uncompromised Intel SGX enabled processor hosting the remote server. Finally, we stress that handling any side-channel attacks against Intel SGX is out of our scope.

2.2 Secure execution inside Intel SGX enclaves

Intel SGX is a hardware assisted mechanism in the form of an ISA extension to the Intel architecture. It is designed to allow secure attestation and sealing to application software executing in a secure environment that is known as *enclave*. The main purpose of these extensions is the protection of selected code parts and data from disclosure or modification in untrusted environments. The enclaves are protected by the CPU that is in charge of any access to the enclave memory or other protected areas of execution. Any instruction that reads or writes to the enclave and is not part of it, fails. Assuming an untrusted or even a malicious operating system, hypervisor or firmware, SGX protects the confidentiality of the enclave pages. An Intel SGX application consists of (i) the untrusted code and (ii) a trusted enclave that it securely calls into. The code and data that are part of the enclave are stored in a DRAM subset, the Processor Reserved Memory (PRM). PRM has a contiguous range and can not be accessed by any system software or peripherals. Moreover, the contents of the enclaves are stored into the Enclave Page Cache (EPC), a subset of PRM. Non-enclave software is not able to access the EPC [15]. For Intel Skylake CPUs, the EPC size is between 64 MB and 128 MB and SGX provides a paging mechanism for swapping pages between the EPC and untrusted DRAM. The data of the enclave that has to be written to the disk is encrypted and checked for its integrity. Between enclaves, SGX enables local attestation. Additionally, in the case of a third party application or software, SGX allows remote attestation to ensure that the application is uncompromised, and therefore can be trusted. SGX enables the remote system to establish a connection with the enclave, using an end-to-end encrypted channel. In Section 4, we present the details regarding the development of TrustAV utilizing the Intel SGX technology.

2.3 Signature-based malware analysis

Signature-based malware analysis is a commonly used technique in state-of-the-art systems. The data under malware analysis are processed against a set of malware signatures in order to identify the presence of malicious software. The Aho-Corasick algorithm is considered as an efficient option for multiple pattern searching, since it matches all signatures in a rule-set simultaneously. For this reason, Aho-Corasick is utilised by popular open source security solutions (e.g. ClamAV antivirus [4], Snort network intrusion detection system [30]). The algorithm constructs a finite state machine that resembles a tree, along with "failure" links between the nodes. Failure links are followed when there is no matching transition, allowing fast transitions to other branches of the tree with a shared prefix and avoiding costly backtracking. To provide a more efficient approach, a Deterministic Finite Automaton (DFA) can be built by unrolling the failure links in advance and adding relevant transitions to map each failure directly to a node without the need to follow multiple failure links at runtime. In Section 4, we present the details of our signature-based malware detection using Aho-Corasick.

3 DESIGN

In this section we describe the design and implementation of the TrustAV architecture. The overview of TrustAV is presented

in Figure 1. The two entities that compose our system are (i) the TrustAV client, which transmits the necessary files to the remote server for scanning, and (ii) the TrustAV server, which is responsible for performing the malware analysis in a privacy-preserving way. The entire malware scanning operation is performed in the cloud-based server, encapsulated inside Intel SGX enclaves. This encapsulation enables the protection of the data processing algorithms, the signature set and most importantly the privacy of the user's data.

3.1 TrustAV Client

The TrustAV client implements three distinct functionalities, using three different components. The client prompts the users to select files or file system regions that they wish to scan and provides a set of actions for each file that is found to be infected. Once the selection is defined, it gathers the data and periodically checks their status by comparing their hash values against known hash values kept in a whitelist. Finally, the client transfers the files whose hash values did not match with the whitelist to the cloud-based TrustAV server which performs the malware scanning and reports the results. The client is required to provide minimum effort and functionality, while the computationally intensive malware scanning is performed by the TrustAV cloud-based server. In this way, the server remains independent from the client implementation, while multiple client versions can be developed to support different platforms and operating systems.

The hashing component is responsible for retrieving the data from the client's file system. The data can be found in the form of various files present in the file system, such as photos, videos, audio files or applications. The client periodically hashes all selected files and directories and forwards the hash values to the whitelist component. The window of the periodic scanning can be defined by each user depending on the device type and current status of each device. The client's whitelist component is responsible for comparing the hash values obtained by the hash computation module against a list of hashes that have been calculated from a known clean state of each file. If the hash values do not match a given file, it is marked as suspicious and the client has to forward it to the cloud-based TrustAV server for malware scanning. Moreover, the client is responsible for the maintenance of the whitelist by managing the entries of new or deleted files and updating existing entries with new benign hash values. The motivation for this periodic hash-checking functionality is threefold. First, calculating and comparing hash values against a set of known hashes—which represent the clean state of files—can be quick and efficient given the plethora of different hash algorithms available. Second, it is a fast preliminary way to filter out benign files from possibly infected ones without having to perform complex malware analysis on every file and application of a device. This allows the TrustAV client to be easily implemented for a wide variety of commodity devices (e.g., desktop, smartphone). Third, by marking only a limited subset of files as potentially infected, we minimize the amount of data that need to be transferred to the remote TrustAV server, improving the overall performance of our system and minimizing the cost for users who perform virus scanning using metered connections, mobile data plans or are connected via a low bandwidth channel.

The most important component of the TrustAV client is the secure communication with the remote server. This component ensures that all the potentially infected files are transmitted to the TrustAV remote server for a thorough malware analysis, in a secure and privacy-preserving manner. Each file is first encrypted using a secret cryptographic key, that is pre-established with the server. After the successful transmission of the marked files, it awaits for the remote server's response. This response is received in an encrypted form, and contains details about the infected files, such as the risk level and the actions to be taken by the user. According to the information provided by the TrustAV server, the client prompts the user with possible actions in order to handle the infected files in the file-system. Then, the whitelist module updates the list for every file whose contents changed in a benign way, thus leading to the generation of a new hash value, and removes the entries of the deleted files.

3.2 TrustAV Server

The server is able to accept connections from multiple TrustAV clients and perform malware analysis on the incoming data. The server maintains an updated signature-set, used for the malware analysis. By keeping the entire signature-set on the remote server, the system is able to benefit in two ways. First, the system does not rely on each user to maintain the latest signature-set locally. Second, a major benefit of offloading the entire malware analysis on the cloud-based TrustAV server is the ability to utilize the Intel SGX enclaves, provided by recent Intel processors. Using SGX enclaves we are able to execute the entire life-cycle of the virus scanning process in a trusted environment, ensuring that any sensitive data obtained by the users, cryptographic keys and signature-sets are never exposed in the server's DRAM or file system. This attribute is crucial for two reasons. First, we can guarantee that users can securely offload sensitive data to the remote server for malware analysis without risking leakages. Second, even if malicious actors manage to compromise the server, they will not be able to identify the signatures used in the signature-set or tamper it in any way. Moreover, Intel SGX enclaves ensure the secure code execution. This makes the virus scanning algorithms immune to attacks while they are executed, preventing code tampering or data leakage from variables in use. Finally, since Intel SGX enclaves operate as a reverse sandbox and the enclave hosting the malware scanning engine only communicates with the client, user data are not accessible even by honest-but-curious providers, hosting the server, ensuring the privacy of the offloaded user data.

The user's data are received in encrypted format by the TrustAV server and are forwarded inside the Intel SGX enclave hosting the engine of our system. Once inside the secure enclave, the data are decrypted and prepared for processing. The cryptographic keys required for the successful decryption of the client's data exclusively reside inside the SGX enclave. In this way, the secret keys and sensitive data, such as personal documents or photos, are never present in plain-text format in the server's file system or DRAM and they remain inaccessible even by the server's host/provider. Moreover, even if the non-SGX part of the TrustAV server or the hosting infrastructure gets compromised, the keys and the private user data can not be obtained.

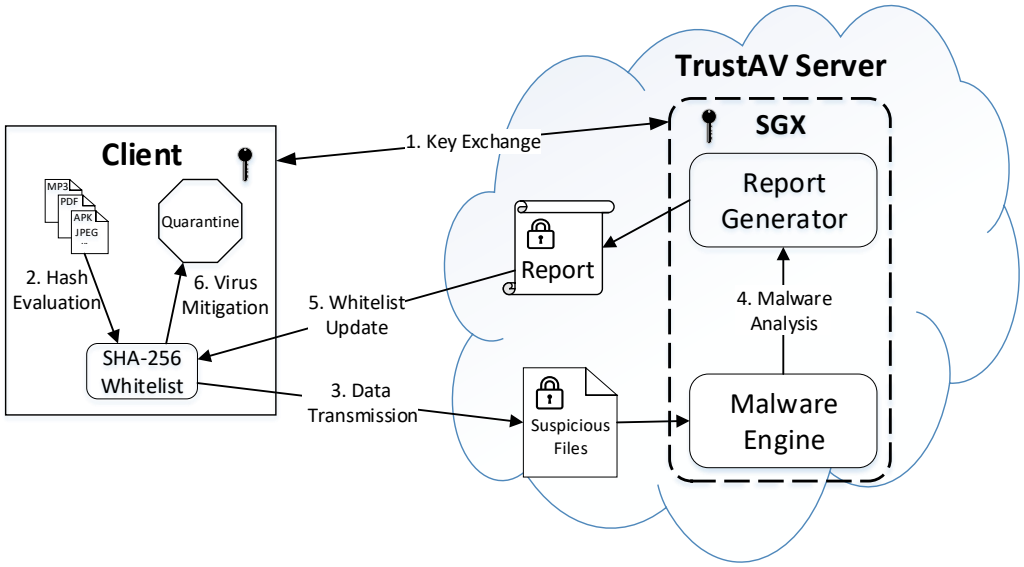


Figure 1: TrustAV design overview.

The malware scanning component constitutes the core of our system and is responsible for processing the incoming data using a given malware signature-set. The entire functionality of this component along with all required data, such as the signature-set, reside only inside the Intel SGX enclave. Each rule is assigned with a unique ID and it is consisted of a set of patterns and information metadata, describing the malware functionality, the risk level and suggested actions. The virus scanning process is performed inside the enclave once the data are decrypted using each client's secret key. When a rule is successfully triggered by an input file, the corresponding metadata along with the file information are processed in order to be forwarded to the client as a status report.

The report generation is performed by a separate component, which also resides inside the secure enclave. During this process, the TrustAV server receives results from the malware scanning engine and generates a report that will be processed by the TrustAV client. This report contains information about the malicious files detected, such as filenames, risk level, scanning date, etc. When the report is constructed, the server encrypts it, while still inside the enclave, and then it is forwarded to the corresponding client. Ensuring that the scanning report never lives outside of the SGX enclaves in plaintext format is very important for the user's privacy. By rendering the report inaccessible outside the user's device or the server's SGX enclave, attackers or honest-but-curious entities, such as the service provider, can not obtain any information about the user's private data. In combination with protecting the signature-set inside the enclaves, we also eliminate the possibility of malicious entities injecting custom signatures and observe the generated report in order to infer information that could threaten the privacy of the user's data. Moreover, the report is randomly obfuscated so that its size can not be used in order to infer information about the number or type of identified malware.

3.3 Service Registration

The registration process is the first task performed by TrustAV when the client is initiated on a user's device. At the first step of this process, the client communicates with the TrustAV cloud-based server and exchanges a shared key. During this process, the server generates a client ID and stores the shared key along with the corresponding ID inside the SGX enclave. The second step is the generation of the list containing the hashes of each file at a clean, uninfected state. For this reason, the client hashes every selected value and temporarily populates the whitelist. Then, every file is transmitted to the remote server for malware analysis. Once the server responds with the report, the hash values of the uninfected files are considered permanent in the whitelist and all the malicious files, if any, are handled by the user according to the report. When the registration process is finished, the TrustAV client prompts the user for a periodic hashing interval and the system defaults in automated periodic scanning.

3.4 Remote Attestation

TrustAV can leverage the Remote Attestation services, provided by Intel in order to further increase the security and level of trust of the SGX-enabled server. Using remote attestation, the TrustAV client challenges the server to verify that the core part of the engine is located inside a signed SGX enclave, executed on an SGX-enabled processor. In this way, we eliminate the possibility of a malicious entity posing as an SGX-enabled TrustAV server in order to obtain access to a user's private data. Moreover, we prevent entities executing the TrustAV server in SGX-debug mode, trying to obtain access to the user's sensitive data and server's secret keys via the use of debuggers.

4 IMPLEMENTATION

In this section, we present the implementation of the applications that compose TrustAV. We provide detailed description of the malware scanning process as well as the steps required to develop an SGX-enabled malware scanning engine, operating on a cloud-based server to protect and preserve the privacy of the user's data.

4.1 Malware Scanning Inside SGX Enclaves

Our malware scanning engine is based on the Aho-Corasick pattern matching algorithm, as it is one of the most efficient and widely used, found in many signature-based solutions, such as the popular open-source ClamAV. TrustAV performs the entire malware scanning process inside Intel SGX enclaves in order to preserve the privacy of the offloaded data, the security of the executed code and the integrity of the signature-set.

As described in Section 2.3, in most implementations, the patterns are compiled into a state machine (DFA) which is constructed as a tree with each node containing information about the state it represents, as well as various metadata. However, this state machine structure is not optimal in order to be used inside SGX enclaves. The reason is twofold: (i) the generated tree requires a lot of memory in order to be represented while the live available memory of SGX enclaves is quite limited, (ii) traversing the nodes located scattered in memory during the pattern matching process eliminates possible caching effects and reduces the sustainable performance. In order to address these constraints that can play a significant role in terms of performance, we choose to represent the DFA as a serialization of the state machine tree to a single-dimensional integer array. First, we compile all the available malware signatures into a single Aho-Corasick DFA. Next, we serialize the produced tree as a two-dimensional integer array. This array consists of 256 columns, which represents the size of the ASCII set (i.e. all the values that a single input byte can take), and N rows, where N is the number of the states in the DFA. Each row represents a DFA state and each cell contains the ID of the next valid transition, corresponding to the ASCII character that the cell represents. In order to traverse the serialized DFA tree, the matching process starts from state 0 (row 0) and selects the appropriate column, according to the ASCII value of the first character of the input. In this cell, it finds the next valid state, which is located in another row of the array. Then, it fetches the next character from the input and moves to the cell pointed to by the row given in the previous step and the column given by the ASCII value of the current character. The final states in the array are annotated with a negative sign. When the task hits a negative state, a match has been successfully found. Then, the search is continued using its absolute value for the next step. The fail states either point the matcher to a previous valid state or to the initial state 0. In practice, as we stated earlier, this array is single-dimensional and all the rows that we mentioned in our example are concatenated. Since the size of every row is 256 integers, the matcher traverses the array as follows: *state = dfa[state * 256 + current_input_byte]*. For this reason, the serialization of the state machine provides a second important benefit to our system. The malware scanning function requires only a few lines of code, rendering it very fast and efficient but, most importantly, produces a minimal Trusted Computing Base (TCB) which is very easy to audit and eliminate any security threatening software bugs.

4.2 SGX Enclave I/O

With the malware scanning process and the signature-set secured inside SGX enclaves, the second –and most important– part of our implementation is to provide secure and efficient I/O with the enclave. The only entry point offered by the enclave is utilized for user-data entry. Since the SGX enclaves do not have access to system calls, the network sockets, necessary for receiving client data, are managed by the non-SGX enabled part of the TrustAV server. The data arrive encrypted via the network while the secret keys required for their decryption exclusively reside inside the SGX enclave. Entering the enclave is achieved via the use of an `Ecall` function. However, multiple consecutive calls to this function can impose a performance overhead to the system. For this reason, we bach incoming encrypted client data using a buffer in the non-SGX enabled part of the application and transfer the buffer into the enclave once it is full. The size of this buffer can be optimized dynamically according to the current workload. Once a batch of user data is gathered, it is forwarded into the enclave for processing.

Before the malware scan can take place, the matcher decrypts the user data with the corresponding key, inside the secure enclave. This ensures that while a compromised server can block the data forwarding into the enclave, the data and the secret keys never reside in main memory or the file-system in plain-text format. The malware analysis results are compiled as a report that can be then parsed by the client. This report contains information about the infected files and the identified malware as well as recommended actions that a user could perform to mitigate each threat. The only data output point of the enclave is utilized in order to transmit the report back to the TrustAV client. Once again, the report is encrypted inside the SGX enclave using the client's secret key. This action is performed in order to guarantee that external observers will not be able to gain any useful information that could disclose the user's file contents or types. After its successful encryption, the report is forwarded to the non-SGX enabled part of TrustAV server in order to be transmitted the client via the network. As an extra privacy preserving guarantee, the report is generated each time with an arbitrary size in order to prevent attackers to infer information by monitoring the report's size.

4.3 Performance Optimizations

One of the biggest challenges of modern signature-based solutions (such as Snort, ClamAV, and TrustAV) is the memory footprint of the signature automata. Usually, the performance of applications that utilize multi-pattern matching algorithms, such as Aho-Corasick, is limited by the cache size provided by the CPU. Once the signature automaton exceeds the cache size, cache misses can greatly hinder the system's performance —a problem that only gets worse as the automaton increases in size when new rules are added. This challenge reaches a new dimension when such an antivirus engine is designed to be executed inside secure SGX enclaves. During execution, enclave code and the required data are placed in a special memory region called Enclave Page Cache (EPC). This

region is protected by being encrypted by a dedicated chip called Memory Encryption Engine (MEE). In this way, memory pages are only decrypted inside the physical processor core while external reads on the memory bus can only observe encrypted data. The EPC size is set by the BIOS and can reach up to 128MB. The SGX driver for the Linux platform supports page swapping, allowing SGX to remove pages from the EPC and place them encrypted in unprotected memory as well as restore them when they are referenced. Pages can not be removed until all cache entries referencing them have been removed from all processor logical cores. This property of the SGX driver provided for Linux allows the allocation of the memory required to store automata exceeding 128MB. However, the random accesses in the automaton's states during the matching process produce a substantial number of accesses to pages stored in unprotected regions, triggering the expensive process of restoring such pages —an overhead that is being added to the one already introduced by the increased number of cache misses as at this point the automaton size is greater than the cache size by orders of magnitude. During the preliminary testing of our system we discovered that this issue becomes very prominent when more than 50% of the input data are infected with malware. Similar behaviour deriving from EPC size is also observed in the literature [9]. Moreover, the SGX driver for Windows does not support this swapping functionality, meaning that automata greater than 128MB can not be stored inside enclaves at all. Considering that automata containing 5000 patterns exceed 128MB, even after serialization, EPC size is a strong limiting factor for porting our application to Windows as well as for every signature based solution that wishes to utilize SGX enclaves.

In order to address this constraint, we develop two custom caching systems, aiming to minimise accesses to pages stored out of EPC as well as enable the protection of big automata for the Windows platform without sacrificing security. Our first approach is a simple cache with configurable size that is limited to 90MB, allowing other data and code to be stored inside the EPC without triggering swapping. This cache stores the first n automaton states while the rest are stored in untrusted memory. In order to protect this part of the automaton, we encrypt each individual transition separately using AES-GCM as provided by the SGX SDK. This process is performed using SGX enclaves for the entire automaton during its compilation, enabling the reconfiguration of the cache size, while the keys required for decrypting the transitions are exclusively stored in the enclave. The SGX enclave is able to access untrusted memory with minimal overhead. During execution, if a transition is not found in the cache, the matcher fetches a copy of its encrypted counterpart, decrypts the contents inside the enclave and proceeds with the malware scanning. In this way, we eliminate the need of costly secure swapping of EPC pages an enable the use of big protected signature-sets for Windows. As we describe in Section 5, this caching system is very efficient due to the fact that according to our micro-benchmarks, in most cases, caching 25% of the automaton results to an average of 85% cache hit ratio when the data are 100% infected. Utilizing the same encryption scheme, we further optimize the caching system by replacing the simple mechanism with an LRU cache. Our goal at this step is to further improve the cache hit ratio while decreasing the cache's memory footprint in order to minimise CPU cache misses and possibly enable the simultaneous operation of several caches, serving different rule-set automata. In order to achieve this, we implement the LRU as a double-linked list with each node holding information about the cached transitions. LRU look-ups are performed using a hash table with each entry being a pointer to a queue node in order to minimise the memory footprint. Finally, we eliminate constant memory allocations when transitions are inserted or evicted from the cache by implementing a memory pool that performs the required allocations during initialization.

4.4 TrustAV Clients

Aiming to cover the vast majority of devices and platforms, we implement both a desktop and a mobile client. The desktop version is targeted for traditional desktop/server devices while the latter is developed as a standard Android APK, that can be utilized by smartphones, tablets and Android-enabled IoT devices. Both clients perform the service registration with the server, as described in Section 3. The hashing of the selected files is performed using SHA-256, utilizing the appropriate libraries offered by each platform. Moreover, the clients implement secure persistent storage of the white-list by exporting it to the file-system encrypted. The exported white-list is also paired with checksums in order to ensure its integrity. The main difference between the two clients is the fact that the desktop implementation is able to utilise remote attestation in order to further enhance the security of the connection with the remote server. This functionality is not available for the Android platform but is not a strong requirement for the execution of our system. Finally, the mobile client offers several options for the fine grained configuration of the scanning intervals in order to minimise network traffic when the device is using a metered connection and optimise battery consumption. Clients targeting other mobile or desktop platforms can be easily developed by implemented the described functionality.

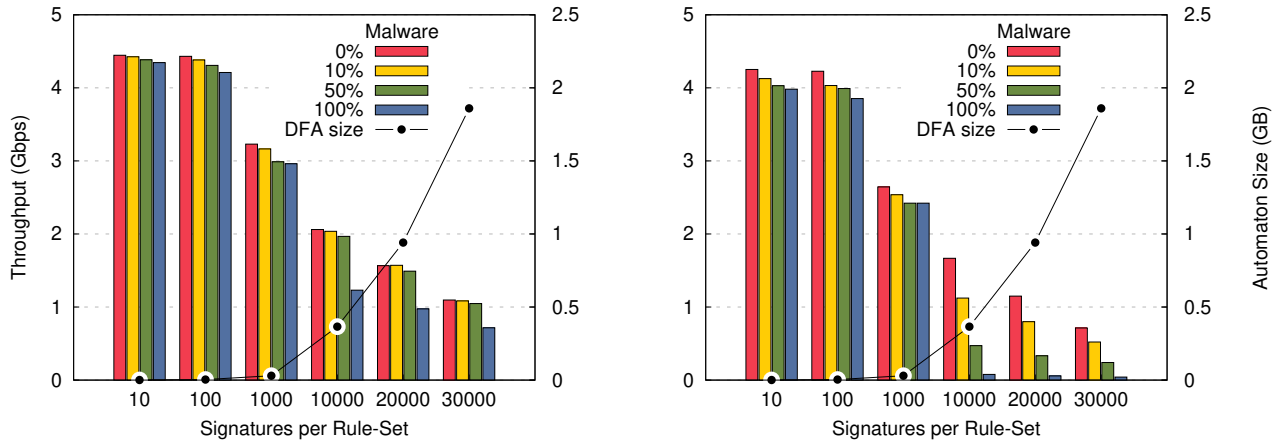
5 EVALUATION

In this section we present the evaluation of TrustAV. First, we analyze the performance characteristics of our malware scanning engine and explain the performance overhead introduced by the usage of hardware assisted enclaves (i.e. Intel SGX enclaves). Then, we evaluate the performance of our caching systems and present their effectiveness compared to default SGX page management.

5.1 Evaluation setup

Hardware Setup. The TrustAV server is hosted on an commodity desktop, based on an eight-core Intel i7-8700K CPU, running at 3.7GHz, providing support for Intel SGX enclaves. The system is also equipped with 32GB of DDR4 RAM clocked at 2400MHz.

Malware Signatures and Workloads. We evaluate the performance of our system using malware signatures utilised by ClamAV. Specifically, we generate sets that contain a varying number of randomly selected signatures (i.e. 10, 100, 1000, 10000, 20000 and 30000), extracted by ClamAV's malware signature database. Each set is compiled into a separate automaton, as described in Section 4. In addition, we generate four different input streams for each signature-set, containing 512MB of data. These streams are composed by various files, injected with signatures so that they report 0%, 10%, 50% and 100% matches. In order to stress the limits of TrustAV, all input



(a) Vanilla (b) SGX Enabled

Figure 2: Throughput evaluation of our malware scanning engine when executed with and without Intel SGX enclaves. Six different signature sets are tested against custom input infected by 0%, 10%, 50% and 100%.

streams that contain matches are crafted so that every signature of each corresponding set is present in the data and every automaton state is accessed.

5.2 Malware Analysis

Figure 2(a) illustrates the sustained throughput achieved by our malware analysis system when executed without using Intel SGX enclaves, while Figure 2(b) presents the performance, when TrustAV is entirely executed within secure enclaves. In addition, each automaton used in each execution is located inside the enclave memory space. Comparing the results, we can see that securing the execution within SGX enclaves adds no more than 19% overhead in execution time when the signature set is able to entirely fit in EPC. This overhead occurs due to (i) the execution of the added CPU instructions, (ii) the accesses inside the enclave's encrypted memory and (iii) the time required to enter and exit an enclave. On the other hand, we can see that the overall execution throughput is reduced for automata exceeding the EPC size when the input streams are infected by 50% or more. In such cases, the multiple frequent random accesses to every automaton state produce a high number of accesses to EPC pages stored in untrusted memory. This behaviour triggers the costly process of restoring them in EPC and evicting others that might be shortly needed after processing only a few bytes from the input. Despite the fact that this issue is not observed with low input stream infection rates, it should be addressed as it severely affects highly infected input data streams by reducing the overall performance by an order of magnitude.

In order to identify the performance penalty introduced by protecting the signature automata, storing them inside the SGX enclave, we re-evaluate the throughput of our malware scanning engine with a different setup. In this case we execute TrustAV server with SGX support but we store the automata in unprotected memory in plain-text format. Since SGX enclaves have full access to unprotected memory with minimal overhead, our engine is still able to process the input streams without exposing the user's data which are still secured and processed inside the enclave. The results of this analysis are presented in Figure 3. Comparing the sustained throughput achieved with automata that exceed EPC size against the performance reported in Figure 2(b), we notice that with this setup the overall performance is significantly increased. This benchmark also provides us with useful information for the implementation and evaluation of the caching systems, described in Section 4.3, as this is the maximum performance that could theoretically be achieved if the EPC page swapping is minimized by 100%. If the TrustAV hosting facility can be completely trusted and the integrity of the signature-sets can be guaranteed or in cases where system performance is a strong requirement, this setup can still be utilized as user data are still never exposed out of the secure enclaves. However, it is not the optimal scenario as potential tampering of the signature set is possible and in such cases frequent integrity checking of the automata should be performed as a minimum countermeasure.

5.3 Performance Optimizations

In this section, we evaluate the performance of the two caching systems that we implement in order to address the performance penalty introduced by severe EPC page swapping, discussed in Section 5.2. By analysing our system's performance with the automaton protected inside the secure enclaves as well as stored in untrusted memory in plain-text format, we are able to identify the introduced overhead for protecting the automaton and the possible maximum performance.

Aiming to preserve TrustAV's security and privacy guarantees, we explore the possibility of encrypting each automaton state using AES-GCM, provided by the Intel SGX SDK, and storing the automaton in unprotected memory in cypher-text format. In this setup, every referenced state is only decrypted inside the SGX enclave during processing and remains protected but without being allocated in EPC pages. We evaluate this setup for automata exceeding EPC capacity using input streams containing only infected files and present the results in Figure 4. While this setup eliminates the need

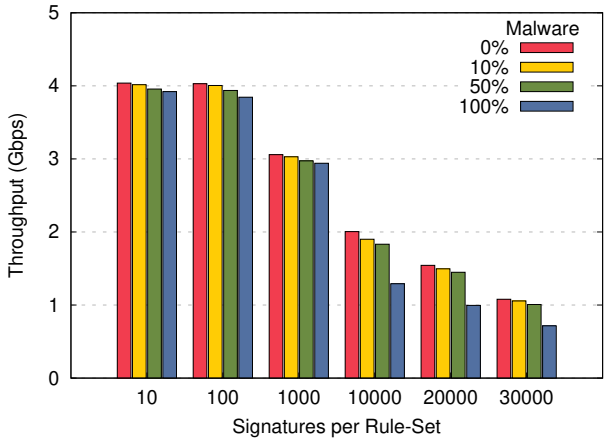


Figure 3: Performance evaluation of TrustAV when executed within SGX enclaves but the automata are stored in untrusted memory in plain-text format.

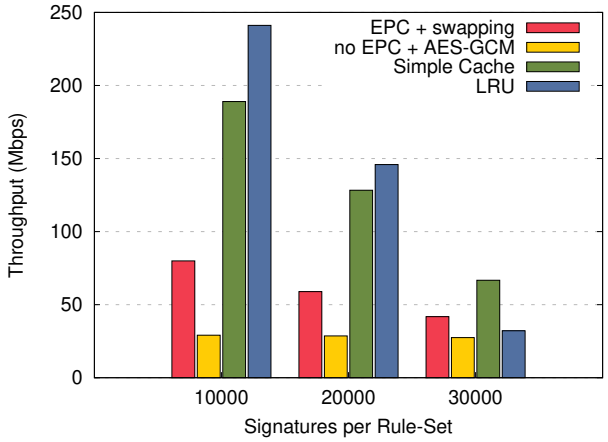


Figure 4: Performance comparison of the four ruleset protection schemes when the automaton size exceeds ECP capacity and the dataset is infected at 100%.

for EPC page swapping, we notice that the performance is lower compared to storing the automaton inside the enclave and letting the SGX driver perform EPC page swapping. The main reason for this is that the decryption of each state using AES-GCM is slower than EPC page swapping, which is backed by the MEE implemented in hardware. Moreover, we notice that this is also the worst case performance for a cache implementation with minimum complexity, operating at 100% cache misses. After exploring the theoretical maximum and minimum performance boundaries of our system, we implement the two caching systems described in Section 4.3 and present their performance evaluation in Figure 4. For this evaluation we use the three signature automata that exceed the EPC memory limits, processing the input streams containing only infected files that produce the worst memory access patterns in the automata. In the simple cache setup, we store up to 90MB of each automaton in EPC, leaving enough memory for the application code, the batches of input data and the malware report generation process. The rest of the automaton is stored in cypher-text format in unprotected memory. Upon a cache miss, the state is fetched from the untrusted memory, decrypted inside the enclave and discarded after usage. We notice that this setup eliminates EPC page swapping while at the same time reduces the accesses to encrypted states in untrusted memory as it offers high cache hit rate, especially for the automata containing 10000 and 20000 patterns. More specifically, we notice that for the used rule-sets, caching 25% of the automaton yields 85% cache hit ratio, meaning that performance can be significantly improved by caching only a small portion of the automaton, as shown in Figure 4. However, this performance benefit gradually decreases as the automaton increases in size and lower percentages of its states can be stored without violating EPC memory limits.

We conclude our evaluation by presenting the sustained throughput achieved by replacing the simple caching system with an LRU cache and executing TrustAV with the same automata and input streams as described above. As we can see in Figure 4, the LRU cache yields the best results for the automata containing 10000 and 20000 patterns by further increasing the cache hit rate, efficiently utilising the available EPC space. However, we notice that it can not outperform the simple cache when processing the input stream using the automaton containing 30000 patterns. The reason for this behavior is that with the given EPC memory limit, the LRU can not store enough states in order to provide substantial cache hit rate increase, compared to the simple cache, while LRU cache misses are more expensive since at each cache miss a state has to be evicted and a new one has to be stored, constantly updating the LRU data structures.

6 RELATED WORK

As expected, the research community in the area of security focuses on malware detection to protect users and organizations from cyber threats that are continually evolving. ClamAV is the most popular open-source anti-malware solution that speeds up scanning and matching [4]. CloudAV was one of the first works to put forward the notion of cloud-based malware scanning while [25] extends CloudAV to the mobile environment. Although CloudAV achieves high detection rate, it exposes sensitive information without preserving the users privacy [24]. SplitScreen implements a distributed anti-malware system to speed up the malware scanning using bloom filters [13]. RScam is another cloud-based anti-malware system which provides efficient security service and data privacy protection for resource-constrained devices [38]. Still, RScam assumes a trusted server environment. In this work, we propose a practical cloud-based malware detection engine using hardware assisted enclaves to shield user data and preserve their privacy. CloudNet is a GPU-accelerated anti-malware engine for cloud services [18]. While some works focus on improving the performance of malware detection systems (e.g. [44]), in this paper we tackle the urge for strong privacy preserving guarantees when it comes to computational offloading to untrusted environments, where users have no control over the manipulation of their personal data. Handheld devices contain a large percentage of personal data, as for instance pictures or transcripts, making them a promising target

for frauds [31, 37]. Some of the most popular commercial applications on mobile malware detection are the AVG Antivirus [1] and Avira mobile security [2] solutions. Google provides its own solution, namely Google Play Protect, as a built-in application that automatically scans and verifies Android applications installed [5]. Unlike TrustAV, this solution is destined only for Android applications retrieved from Google's Play Store. Similar research works based on signature scanning, propose solutions that aim for low power consumption [19, 21]. Other techniques for malware detection are code and API call analysis, like in ScanMe Mobile [48] and [7], respectively. In Paranoid Android [28], security checks are applied on remote security servers that host exact virtual replicas of devices, applying multiple detection techniques simultaneously. Other works, utilize machine learning techniques to apply malware detection on mobile devices [8, 47]. Finally, Deyannis et al. propose a GPU-assisted antivirus solution on Android devices through edge offloading [16]. While our work is based on the same grounds (i.e. the malware detection with support for mobile devices), we advance the state-of-the-art by proposing a practical, yet secure, malware detection engine that strongly focuses on preserving user privacy, offloading the processing of personal data inside a trusted execution environment.

As the need for lower costs, higher performance and scalability rises, outsourcing network processing applications to the cloud has become tempting. APLOMB is a service for outsourcing enterprise middlebox processing to the cloud [34]. To provide confidentiality, BlindBox proposes the processing of encrypted traffic [35], something that leads to unpractical computational requirements. Likewise, Embark, aiming to offer security and confidentiality, enables a cloud provider to support middlebox functionality by processing encrypted traffic [20]. A common service that cloud-based services providers offer is storage, yet, there is no transparency over the manipulation of user data [22, 43, 45, 46]. CloudFence provides transparent data tracking capabilities to both service providers and users [26]. As already mentioned, TEEs, such as Intel SGX, can guarantee data and code protection. Thus, recently, a significant number of works focus on proposing the exploitation of this technology for outsourced applications in the cloud. For instance, VC3 [33], Opaque [49] and [14] offer privacy preserving data analytics in the cloud using Intel SGX. EnclaveDB [29] is a database engine that can guarantee confidentiality, integrity, and freshness for data and queries. In addition, EndBox [17], ShieldBox [41] and SafeBricks [27] focus on securing middlebox functionality using Intel SGX. Unlike these works, our system aims to serve a cloud-based anti-malware solution for any device type respecting their diverse requirements. Finally, while there are works that enable the execution of unmodified applications in enclaves (e.g. [9, 10, 36, 39, 42]), we choose not to follow such approach (i.e. execute our antivirus solution on top of SGX using one of the aforementioned tools) since these tools result to an increased trusted computing base (TCB), which widens the attack surface [11, 32].

7 LIMITATIONS AND FUTURE WORK

Our proposed TrustAV cloud-based malware scanning solution is based on a pattern matching technique in order to identify infected data. As discussed in the paper, such approaches can face performance limitations when the automata required for the data processing are not able to entirely fit in secure enclaves and the input data are highly infected. Also, it is possible that, in certain applications, our TrustAV server might operate with more than 30000 virus signatures in total. Depending on the use-case, preliminary testing should be applied in order to identify the properties of the automaton as different rules might generate automata with different caching characteristics while the access patterns on each automaton depend on the nature of the input data. As future work we plan to extend our system so that the server can identify which rules should be applied on each input stream. In this way, the server will be able to utilize subsets of the rule-set so that smaller automata will be generated, providing smaller memory footprints and better caching properties. We expect that this technique will further improve the overall performance of our system, regardless of the infection rate of the input data.

For the evaluation of our proposed system we utilize real malware signatures, extracted by the database of the popular open-source antivirus ClamAV. The input data streams are synthetic but contain data that trigger all the malware signatures used. We chose to evaluate our systems using synthetic input in order to be able to control the infection rate as well as provide input that triggers the worst possible access patterns in the automata. In this way we can stress the limits of our system and provide a thorough evaluation. In the future, we also plan to evaluate the end-to-end performance of our system using real input streams collected by various platforms.

It is true that side-channel attacks are proven to be feasible on SGX enclaves. However, protecting SGX enclaves from side-channel attacks that either focus on software or hardware bugs is orthogonal to TrustAV and thus we consider that it is out of scope of our work. However, any successful attempt to protect SGX-enabled code/hardware has a direct benefit to our system and we plan to integrate such protections in the future, should they become available.

8 CONCLUSION

In this work, we propose TrustAV, a practical and privacy preserving cloud-based malware analysis solution. TrustAV offloads the intensive process of malware analysis to a remote server with Intel SGX support to protect offloaded personal data as well as the scanning execution against untrusted parties. TrustAV is capable to perform with a minimum performance overhead, which is introduced by the utilization of hardware assisted enclaves. To reduce the memory footprint of our implementation, an important limiting parameter for the majority of equivalent solutions in the state-of-the-art, we develop a caching scheme that eliminates EPC memory swapping and offers up to *3x* speedup.

9 ACKNOWLEDGMENTS

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 2767). This work was also supported by the projects CONCORDIA, I-BiDaaS and C4IIoT, funded by the European Commission under Grant Agreements No. 830927, No. 780787 and No. 833828. This publication reflects the views only of the authors, and

the Commission cannot be held responsible for any use which may be made of the information contained therein.

REFERENCES

[1] [n.d.]. *AVG AntiVirus for Android.*

[2] [n.d.]. Avira: Download security, privacy, and speed-enhancing apps for Android and iOS. https://www.avira.com/en/mobile-security.

[3] [n.d.]. Cisco Visual Networking Index: Forecast and Trends, 2017-2022. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[4] [n.d.]. ClamAV | Cisco Talos Intelligence Group. https://www.talosintelligence.com/clamav.

[5] [n.d.]. Google Play Protect. https://www.android.com/play-protect/.

[6] [n.d.]. Monthly Threat Report (March 2019), Symantec. https://www.symantec.com/security-center/publications/monthlythreatreport.

[7] Yousra Aafer, Wenliang Du, and Heng Yin. 2013. Droidapiminer: Mining api-level features for robust malware detection in android. In *International conference on security and privacy in communication systems*. Springer, 86–103.

[8] Vitor Monte Afonso, Matheus Favero de Amorim, André Ricardo Abed Grégio, Glauco Barroso Junquera, and Paulo Lício de Geus. 2015. Identifying Android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques* 11, 1 (2015), 9–17.

[9] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Daniel O'Keeffe, Mark L Stillwell, et al. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation*.

[10] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 8.

[11] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2018. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel {SGX}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1213–1227.

[12] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. 2014. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM.

[13] Sang Kil Cha, Iulian Moraru, Jiyong Jang, John Truelove, David Brumley, and David G Andersen. 2010. SplitScreen: Enabling Efficient, Distributed Malware Detection.. In *NSDI*. 377–390.

[14] Swarup Chandra, Vishal Karande, Zhiqiang Lin, Latifur Khan, Murat Kantarcioglu, and Bhavani Thuraisingham. 2017. Securing data analytics on sgx with randomization. In *European Symposium on Research in Computer Security*. Springer.

[15] Victor Costan and Srinivas Devadas. [n.d.]. *Intel SGX explained.* Technical Report. Cryptology ePrint Archive, Report 2016/086, 2016.

[16] Dimitris Deyannis, Rafail Tsirbas, Giorgos Vasiliadis, Raffaele Montella, Sokol Kosta, and Sotiris Ioannidis. 2018. Enabling GPU-assisted antivirus protection on android devices through edge offloading. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 13–18.

[17] David Goltzsche, Signe Rüsch, Manuel Nieke, Sébastien Vaucher, Nico Weichbrodt, Valerio Schiavoni, Pierre-Louis Aublin, Paolo Cosa, Christof Fetzer, Pascal Felber, et al. 2018. EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 386–397.

[18] George Hatzivasilis, Konstantinos Fysarakis, Ioannis Askoxylakis, and Alexander Bilanakos. 2018. CloudNet Anti-malware Engine: GPU-Accelerated Network Monitoring for Cloud Services. In *International Workshop on Information and Operational Technology Security Systems*. Springer, 122–133.

[19] Hahnsang Kim, Joshua Smith, and Kang G Shin. 2008. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM, 239–252.

[20] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. 2016. Embark: securely outsourcing middleboxes to the cloud. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*.

[21] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. 2009. Virusmeter: Preventing your cellphone from spies. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 244–264.

[22] Nick Nikiforakis, Marco Balduzzi, Steven Van Acker, Wouter Joosen, and Davide Balzarotti. 2011. Exposing the Lack of Privacy in File Hosting Services.. In *LEET*.

[23] Jon Oberheide, Evan Cooke, and Farnam Jahanian. 2008. CloudAV: N-version Antivirus in the Network Cloud. In *Proceedings of the 17th Conference on Security Symposium (SS'08)*.

[24] Jon Oberheide, Evan Cooke, and Farnam Jahanian. 2008. CloudAV: N-Version Antivirus in the Network Cloud.. In *USENIX Security Symposium*. 91–106.

[25] Jon Oberheide, Kaushik Veeraraghavan, Evan Cooke, Jason Flinn, and Farnam Jahanian. 2008. Virtualized in-cloud security services for mobile devices. In *Proceedings of the first workshop on virtualization in mobile computing*. ACM, 31–35.

[26] Vasilis Pappas, Vasileios P Kemerlis, Angeliki Zavou, Michalis Polychronakis, and Angelos D Keromytis. 2013. CloudFence: Data flow tracking as a cloud service. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 411–431.

[27] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. SafeBricks: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*.

[28] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. 2010. Paranoid Android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 347–356.

[29] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database using SGX. In *EnclaveDB: A Secure Database using SGX*. IEEE, 0.

[30] Martin Roesch et al. 1999. Snort: Lightweight intrusion detection for networks.. In *Lisa*, Vol. 99. 229–238.

[31] Brendan Saltaformaggio, Rohit Bhatia, Zhongshu Gu, Xiangyu Zhang, and Dongyan Xu. 2015. Vcr: App-agnostic recovery of photographic evidence from android device memory images. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 146–157.

[32] Vasily A Sartakov, Stefan Brenner, Sonia Ben Mokhtar, Sara Bouchenak, Gaël Thomas, and Rüdiger Kapitza. 2018. EActors: Fast and flexible trusted computing using SGX. In *Proceedings of the 19th International Middleware Conference*.

[33] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy*.

[34] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 13–24.

[35] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blindbox: Deep packet inspection over encrypted traffic. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 213–226.

[36] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. 2017. Panoply: Low-TCB Linux Applications With SGX Enclaves.. In *NDSS*.

[37] Junliang Shu, Yuanyuan Zhang, Juanru Li, Bodong Li, and Dawu Gu. 2017. Why data deletion fails? A study on deletion flaws and data remanence in Android systems. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 2 (2017), 61.

[38] Hao Sun, Xiaofeng Wang, Jinshu Su, and Peixin Chen. 2015. Rscam: Cloud-based anti-malware via reversible sketch. In *International Conference on Security and Privacy in Communication Systems*. Springer, 157–174.

[39] Hongliang Tian, Yong Zhang, Chunxiao Xing, and Shoumeng Yan. 2017. SGXKernel: A Library Operating System Optimized for Intel SGX. In *Proceedings of the Computing Frontiers Conference*. ACM, 35–44.

[40] Eran Toch, Claudio Bettini, Erez Shmueli, Laura Radaelli, Andrea Lanzi, Daniele Riboni, and Bruno Lepri. 2018. The privacy implications of cyber security systems: A technological survey. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 36.

[41] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2018. ShieldBox: Secure Middleboxes using Shielded Execution. In *Proceedings of the Symposium on SDN Research*. ACM, 2.

[42] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC)*.

[43] Marten Van Dijk, Ari Juels, Alina Oprea, Ronald L Rivest, Emil Stefanov, and Nikos Triandopoulos. 2012. Hourglass schemes: how to prove that cloud files are encrypted. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 265–280.

[44] Giorgos Vasiliadis and Sotiris Ioannidis. 2010. Gravity: a massively parallel antivirus engine. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 79–96.

[45] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. 2010. Privacy-preserving public auditing for data storage security in cloud computing. In *2010 proceedings ieee infocom*. Ieee, 1–9.

[46] Jia Xu, Ee-Chien Chang, and Jianying Zhou. 2013. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 195–206.

[47] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. 2014. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *European symposium on research in computer security*. Springer, 163–182.

[48] Hanlin Zhang, Yevgeniy Cole, Linqiang Ge, Sixiao Wei, Wei Yu, Chao Lu, Genshe Chen, Dan Shen, Erik Blasch, and Khanh D Pham. 2016. ScanMe mobile: a cloud-based Android malware analysis service. *ACM SIGAPP Applied Computing Review* 16, 1 (2016), 36–49.

[49] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform.. In *NSDI*. 283–298.