A Secure Network Stack for the Untrusted Cloud

Keita Aihara*
k.k.ahihara@sslab.ics.keio.ac.jp
Keio University, Japan

Pierre-Louis Aublin
pl@sslab.ics.keio.ac.jp
Keio University, Japan

Kenji Kono
kono@sslab.ics.keio.ac.jp
Keio University, Japan

Introduction. *Trusted execution* is indispensable to service providers that deploy their applications on third-party computing platforms such as Amazon EC2, Google Cloud, or Microsoft Azure. Service providers expect confidentiality and integrity of their applications to be guaranteed by platform providers. Although platform providers are not inherently malicious, they are a target of attacks yet securing their computing platforms is a difficult task. The unfortunate consequence is frequent data leakage or corruption of Internet services.

Recent years have seen the development of Trusted Execution Environments (*TEE*), in particular with Intel SGX, ARM TrustZone or AMD SME/SEV. TEEs are special secure execution environments isolated from the rest of the system, including privileged software. By leveraging a TEE, service providers can protect their applications from potentially malicious privileged software; and platform providers can guarantee stronger security to their customers.

Many TEE-based applications have been proposed in the recent years. Some of these systems embed a full network stack inside the TEE, primarily for performance reasons and ease of use. However they all rely on the untrusted host operating system to provide communication capabilities.

The usage of an untrusted TCP stack has two negative consequences: (i) no isolation from other untrusted components (kernel or application); and (ii) no protection of the communication metadata. First, to limit the impact of bugs, security breaches or data leakage, it is of prime importance to isolate the TCP stack from the kernel and from the applications. Second, existing protocols such as TLS or IPsec do not guarantee the integrity or confidentiality of the metadata. This is a problem, as an attacker could learn sensitive information from the metadata, despite the data being encrypted.

Our proposal. To tackle this problem we describe *Shinkansen*, a Secure TCP/IP stack design for commodity machines that provides integrity and confidentiality guarantees from a malicious software stack from the network card to the final application by leveraging a TEE. Shinkansen replaces the untrusted network stack with a trusted network stack and uses a special NIC with encryption functionalities. This can be implemented (but not required) by a smartNIC or FPGA: our goal is to minimize the amount of processing at the NIC to facilitate our system adoption. Shinkansen furthermore proposes a simple API to the final application for user-specific computation such as debugging, auditing or data exfiltration prevention.

Shinkansen meets the following objectives:

1) *Security of communications*: Shinkansen ensures the integrity and confidentiality of both the data and metadata.

2) *Isolation*: the TCP stack and application are executed in two different protection domains.

3) *Performance*: Shinkansen provides good performance.

4) *Debugging and auditing*: Shinkansen provides an API for application-specific debugging and auditing.

Execution flow. IP packets are received by a *network card with encryption functionalities*. It signs, encrypts and shares the packets with the main memory. A fast *user-space packet I/O library* forwards them to the TCP stack. It does not involve the operating system nor inspects the packets during communications. A *secure network stack*, executing inside a TEE, processes the IP packets, recreating the TCP stream. It can furthermore execute application-specific processing for debugging or auditing purposes. The TCP stream is finally exchanged with the application (executing in its own enclave) via TCP buffers allocated in untrusted memory. As we assume the application uses a secure communication protocol such as TLS, the content of the TCP buffers is protected both for confidentiality and integrity.

Application specific API. Shinkansen secure network stack provides a stream and packet processing API for application specific processing. It targets 3 classes of functionalities: (i) debugging; (ii) logging; and (iii) security. Application developers can use this API to implement specific processing.

For security reasons the application specific functions cannot interact with code outside of the TEE. This is to prevent code that would extract sensitive information, such as the TCP packets or the plaintext application payload, to the untrusted environment. the code is automatically checked at compile time for the absence of function calls outside the TEE.

Implementation. We have implemented a prototype of our design leveraging Intel SGX, the mTCP user-level TCP stack and the DPDK library. To provide good performance and limit the impact of faults, Shinkansen (i) minimizes the amount of code and data in the TEE; (ii) minimizes the number of transitions between the untrusted and trusted environments; (iii) implements batching of network functionalities; and (iv) implements a novel enclave call delegation mechanism that avoids enclave transitions.

Evaluation. We consider three real-world applications: the Lighttpd web server, the Memcached key-value store and a video streaming service. Using an SGX-capable processor and a 10Gbps network link, our preliminary evaluation demonstrates that Shinkansen offers strong security guarantees while reducing the performance by less than 9%.

* Presenter and student