TAaaS: Trustworthy Authentication as a Service

Jaemin Park
The Affiliated Institute of ETRI
13 PUBLICATIONS **228** CITATIONS
SEE PROFILE

Sungjin Park
The Affiliated Institute of ETRI
5 PUBLICATIONS **1** CITATION
SEE PROFILE

TAaaS: Trustworthy Authentication as a Service Based on Trusted Path

Jisoo Oh, Jaemin Park, Sungjin Park and Jong-Jin Won
The Affiliated Institute of ETRI
Daejeon, Republic of Korea
{*jsoh77, jmpark, taiji, wonjj*}*@nsr.re.kr*

***Abstract*—Authentication as a Service (AaaS) provides on-demand delivery of multi-factor authentication (MFA). However, current AaaS has left out of consideration the trustworthiness of user inputs at client devices and the risk of privacy exposure at the AaaS providers. To solve these concerns, we present TAaaS, Trustworthy Authentication as a Service, which offers a trusted path-based MFA service to the service provider in the cloud. TAaaS leverages the hypervisor-based trusted path to ensure the trustworthiness of user inputs, and addresses privacy concerns in the cloud by storing only the irreversible user account information. We implement two end-to-end prototypes and evaluate our work to show its feasibility and security.**

***Keywords*-trusted path, authentication as a service, cloud computing**

I. Introduction

As cloud computing provides cost-effective and on-demand delivery of computing resources, security is also delivered as a service from the cloud, named Security as a Service (SECaaS). SECaaS provides third party service providers (SP) with the delegation of integrated security infrastructure [1]. Authentication as a Service (AaaS) is one of the SECaaS that offers multi-factor authentication (MFA) on demand. AaaS is considered a desirable cloud-based service because it enables the SP to apply strong authentication with less effort. Several corporations have endeavored to provide improved authentication based on AaaS [2]–[4].

However, AaaS has been still suffering from security threats: *privacy exposure at the AaaS provider* and *untrusted user inputs in client devices*. First, the risk of privacy exposure is a significant concern in the cloud because the nature of multi-tenancy enables that data belonging to different customers can be resided on the same physical machine by a certain resource allocation policy [5]. Although there is a potential risk that the delegated data could be disclosed to the public, the SPs need to entrust AaaS providers with *user account information* that is a set of user identifiers such as the user ID and the hashed password. Second, *authentication data*, the confidential data from authentication factors (e.g., passwords), is input by users via untrusted client devices. Existing legacy operating systems (OSes) do not guarantee the trustworthiness of their execution environments, so the safety of user inputs cannot be assured in the client device. For example, keyloggers could obtain keystrokes, especially regarding authentication data, without user perceptions [6].

Although many studies on AaaS have been undertaken, most efforts focused only on the security and/or privacy of the cloud infrastructures, protocols, and services [7]–[11]. However, to the best of our knowledge, there exists no AaaS that considers both risk of privacy exposure and the untrusted user input together.

In this paper, we propose Trustworthy Authentication as a Service (TAaaS), a novel service model that guarantees the trustworthiness of the authentication data input by users and privacy preservation of the user account information. Our work provides a trusted path between the user input and TAaaS based on the isolated execution environment from the untrusted legacy OS. Remote attestation is also introduced to enable TAaaS to vouch for the trustworthiness of both the user input and the execution environment. On the basis of the established trusted path, TAaaS provides existing SPs with the trustworthy MFA service.

The main contributions of this paper are as follows.

- **AaaS based on trusted path.** We propose the trusted path-based AaaS to assure the trustworthiness of user inputs and to guarantee that no authentication data is leaked to attackers throughout the TAaaS protocol. Thus, TAaaS can securely handle user input even in a compromised legacy OS of the client device.
- **Privacy-preserving AaaS.** To resolve the privacy issue of existing AaaS, the SP delegates user authentication to TAaaS with the irreversible user account information generated via cryptographic hash function. Therefore, no one other than the SP can get the genuine data.
- **Implementation and evaluation of TAaaS.** We port two commonly used applications, Apache HTTP Server (legacy system) and Etherpad (Software as a Service) to show the feasibility of TAaaS.

The rest of this paper contains the followings: we give a brief explanation about background in Section II. Section III describes our goal and adversary models. We propose TAaaS in Section IV, and provide prototype implementation and evaluation of TAaaS in Section V. We discuss limitations of TAaaS in Section VI. Section VII is a review of related work on trusted path and AaaS. We conclude this paper in Section VIII.

II. Background

A. Authentication as a Service

AaaS is designed for easy adoption of authentication services, and provides on-demand delivery of the MFA. It reduces the time and cost of management for the SPs. Moreover, the legacy systems of the SPs are not modified much because the AaaS API enables easy adoption of the target systems. Two types of previous works on AaaS are presented: *delegation* [2]–[4], [9], [10] and *single sign-on (SSO)* [7], [8], [11]. In delegation, the SPs delegate verification of authentication data sent from users by entrusting AaaS with user account information. In contrast, in SSO, AaaS generates authentication tokens depending on the user authentication, and users access the SPs using these tokens.

However, security threats remain with AaaS, such as risk of privacy exposure of the user account information and untrusted user inputs in client devices.

B. Trusted Path

The *trusted path* establishes a secure channel that guarantees the authenticity, and optionally secrecy and availability of data transfers between a software module and a user's peripheral [12]. Without the trusted path, an adversary can obtain user inputs by recording keystrokes. To guarantee the trustworthiness of the trusted path, the verifiable trusted computing base (TCB) should be leveraged. The TCB is the totality of protection mechanisms that should be trusted to achieve the given level of security [13]. It should have a very small code base to reduce the possibility of vulnerability. Thus, the trusted path is usually supported on small footprints TCBs such as the ARM TrustZone-aware OS, hypervisors, etc. Among them, the tiny hypervisor could be the best option because the combination of Dynamic Root of Trust for Measurement (DRTM) and the Trusted Platform Module (TPM) can compute a verifiable eigenvalue of TCB and user inputs.

To establish the trusted path using the hypervisor, remote attestation, which includes measuring and verification, should be performed.

Measuring. The attestation is based on the *measurement* process, which denotes computing a cryptographic hash over the TCB code before it is executed, and DRTM enables the measured launch at any time [14]. The measurement is used to make a security decision about the attesting platform by a remote entity. It must be placed in secure storage like the TPM. The TPM is a hardware security chip designed to improve platform security with protected storage, called Platform Configuration Registers (PCRs). The value of a PCR can be modified by the TPM `Extend` operation which applies the cryptographic hash as follows:

$$\texttt{Extend}\rightarrow\texttt{PCR}n\texttt{=Hash(PCR}n\texttt{||data)}$$

Especially, PCR17 is extended with platform measurement only by privileged instructions like `SKINIT` in AMD processors, enabled by DRTM. Thus, the value of PCR17 can be used to assure the integrity of TCB. In addition, to check the integrity of user inputs as well as the TCB, the user inputs should be extended into another PCR.

Verification. An Attestation Identity Key (AIK) is a TPM-resident RSA key pair, and the private AIK cannot leaves the TPM without wrapping. The TPM `Quote2` operation signs the measurement (PCRs) with the private AIK, and the simplified semantic of `Quote2` is as follows:

$$\texttt{Quote2}\rightarrow\texttt{Sign}_{\texttt{AIK}}\texttt{(Hash(PCRs||nonce||locality))}$$

The locality, the privileged level for the caller-based access control of the TPM requests, is passed to the TPM `Quote2` operation to identify which caller issued this operation. The remote entity verifies the result of `Quote2` using the public AIK certified by the privacy certification authority (CA), and finally it can trust the user inputs from the TCB.

III. Problem Definition

A. Goals

Our goal is to provide Trustworthy Authentication as a Service (TAaaS), which is a trustful MFA service without the delegation of user account information. In this subsection, we present the requirements needed to satisfy the goal.

AaaS based on the trusted path. Even though the AaaS mechanism is secure enough, the authentication data still remains vulnerable to attacks. Adversaries can intercept user input by injecting malwares (e.g., rootkit) into the untrusted legacy OS. Thus, we target to supplement conventional AaaS with a trusted path between the user inputs and TAaaS to protect the authentication data.

Privacy-preserving AaaS. To introduce existing AaaS, the SPs should delegate their user account information to the AaaS providers. However, this delegation may cause the privacy exposure of the account information in the cloud. To mitigate this issue, our service model guarantees that TAaaS does not need to manages any original account information from the SP.

Minimized and verifiable TCB. The TCB should remain as small as possible to reduce the probability of being vulnerable to various attacks. Furthermore, a verifier should be able to check whether the TCB of a target device has been compromised, otherwise it cannot trust the data from the TCB. Therefore, we aim to minimize the TCB size as well as to provide a cryptographic way to attest to the TCB.

B. Adversary Models

We consider two types of adversaries: attacks in the client device, and attacks in the cloud.

Attacks in the client device. These adversaries impersonate a legitimate user with authentication data acquired maliciously. They may compromise the OS, inject keyloggers, or perform hyperjacking [15] to capture the keystroke.

They could also gather authentication data previously used by applications.

Attacks in the cloud. These adversaries are a type of insider attack on the cloud service provider (e.g., administrator, staffs, etc.) who aim to threaten privacy. They can access any user account information stored in the cloud and monitor the parameters passed by cloud APIs.

We assumed that network security protocols such as TLS/SSL were deployed in our service model; thus, we do not consider security threats in the network.

IV. Design

A. Overall TAaaS Model

Figure 1 shows the TAaaS architecture consisting of three components: the client device, the SP, and TAaaS.

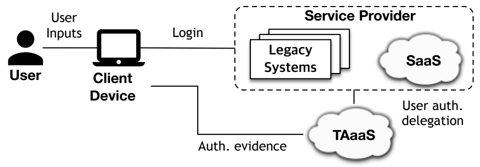


Figure 1: Overall TAaaS Model.

Client device. The users access the SP with their own devices using legacy applications such as web browsers. In our model, the client device equips both the TPM and the DRTM-enabled processor. Moreover, the hypervisor and the agent are software components newly imposed. The hypervisor is measured at boot time via DRTM, and leverages the TPM to establish the trusted path. The agent mediates between the hypervisor and TAaaS by invoking hypercalls, which are software traps from an application to the hypervisor.

Service provider. The SP is a set of legacy systems and Software as a service (SaaS), which enhances its user authentication by adopting TAaaS. When users try to login, the SP requests the user authentication via cloud APIs.

TAaaS. TAaaS is a cloud service that enables the trustworthy MFA via the hypervisor-based trusted path. In this design, our work leverages the one time password (OTP), which is an example of MFA. The OTP seed is assumed to be mapped with the hashed user account information, which is the hashed digest of both the user ID and the hashed password and is initially registered by the SP. We also assume that communication channels between TAaaS and other components are secured by existing network security protocols such as TLS/SSL.

B. TAaaS Protocol

Figure 2 is a sequence diagram that describes the TAaaS protocol in detail.

Step 1 Login initiation. First, the user and the SP start a login process. The legacy application at the client device shows the login page from the SP to the user. The SP invokes `TAaaSInitTP` with the accessed client IP address. `TAaaSInitTP` is a cloud API provided by TAaaS to initiate a hypervisor-based trusted path with the client device.

Step 2 Trusted path establishment. TAaaS generates a nonce for the particular user and stores a tuple consisting of the client IP address, the nonce, and the session into its own internal database. The session is an identifier of the secure communication between TAaaS and the agent. TAaaS activates the hypervisor-based trusted path by sending the nonce to the agent, and then sends the result in response to `TAaaSInitTP`. The agent invokes `hc_start_tp`, which is a hypercall for activating the trusted path.

Step 3 Authentication data concealment. The user starts to enter the ID, the password, and the OTP into the login page, and these are immediately intercepted by the hypervisor. The ID is forwarded to the OS without any manipulation since it is used to identify the user account information at the SP. However, the password and the OTP are replaced with *placeholders* for secrecy. Thus, the legacy application transmits the ID and the placeholders to the SP.

Step 4 Authentication evidence request. After receiving the ID and the placeholders, the SP requests that user authentication be sent to TAaaS via the `TAaaSReqAuth` API with the client IP address and *the hashed digest of the user account information* corresponding to the transferred ID. The hashed digest is used to verify *authentication evidence*, which is the result of the TPM `Quote2` operation. Then, TAaaS requests the agent to send the authentication evidence, and the agent calls `hc_req_evidence` the hypercall to request authentication evidence.

Step 5 Authentication evidence generation. The hypervisor generates the authentication evidence by performing the TPM `Quote2` operation with the user inputs, the nonce, and the private AIK. The detailed generation process is explained in Section IV-C. The agent delivers the authentication evidence to TAaaS along with the AIK certificate.

Step 6 Authentication evidence verification. Finally, to see if the hypervisor and the authentication data are compromised, TAaaS verifies the authentication evidence. The detailed process will be described in Section IV-C. Depending on the verification result, the SP decides to allow the user to access its resources or not.

C. Hypervisor-based Trusted Path

The trusted path is essential to provide the trustworthy user authentication, and the hypervisor is a key component to build the trusted path. The hypervisor manipulates the keyboard input for secrecy and generates the authentication evidence for the integrity. It includes three core modules for the trusted path as depicted in Figure 3: Hypercall Handler (HH), Port Access Handler (PAH), and Authentication Evidence Generator (AEG).

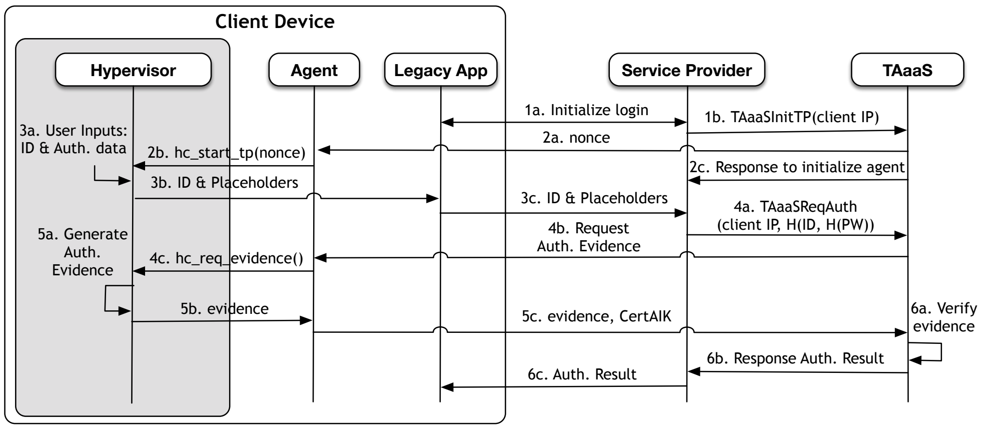


Figure 2: TAaaS Protocol.

Hypercall Handler. The HH processes two hypercalls from the agent: `hc_start_tp` and `hc_req_evidence`.

`hc_start_tp` requests to initiate the hypervisor-based trusted path and passes a nonce from TAaaS to the hypervisor. The nonce is used to guarantee the freshness of the authentication evidence. Therefore, the HH activates the PAH to manipulate keystrokes for establishing the trusted path between the keyboard and TAaaS, and passes the nonce to the AEG for reservation.

`hc_req_evidence` requests the generation of authentication evidence for a particular user whom the SP requests to authenticate, so that the HH executes the AEG.

Port Access Handler. While the PAH is activated, it intercepts every keystroke. When the user presses a key, the keyboard sends a scan code to the output buffer of the keyboard controller. The PAH reads this buffer earlier than the keyboard interrupt handler of the OS, and replaces it with a placeholder as intended.

Figure 4 depicts how the PAH manipulates the user input. When the user enters the ID, the PAH allows the OS to read the original data. In contrast, when authentication data is entered, the PAH replaces the scan codes with placeholders (e.g., asterisk characters). The PAH can distinguish the authentication data from the ID via a tab or semicolon key pressing from the user. Consequently, the placeholders are transmitted to the SP while the genuine authentication data is passed to the AEG for delivery to TAaaS as the form of the authentication evidence. In this way, the secrecy of the authentication data between the keyboard and TAaaS is guaranteed.

Authentication Evidence Generator. The AEG stores the user inputs and the nonce from TAaaS, and generates the authentication evidence. Figure 5 shows the authentication evidence generation at the AEG and the verification at TAaaS. The following sequence describes the procedure of authentication evidence generation:

1) At boot time, the measurement of the hypervisor is extended into PCR17 by the DRTM instruction (i.e., SKINIT).
2) When TAaaS requests an authentication evidence to the agent, the agent calls `hc_req_evidence`.
3) The AEG extends the user inputs. The ID and the hashed password are extended into PCR21, and the OTP is extended into PCR22.
4) The AEG signs information about the integrity of the user input and TCB with the private AIK using the TPM `Quote2` operation. This information indicates PCR17, PCR21, PCR22, the nonce, and the locality.

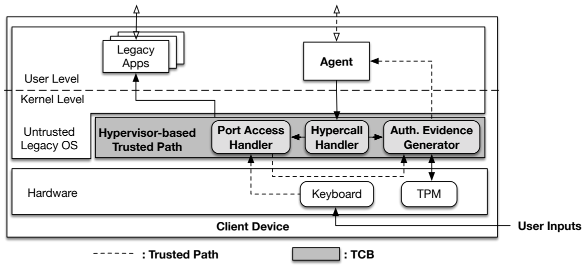


Figure 3: Hypervisor-based Trusted Path.

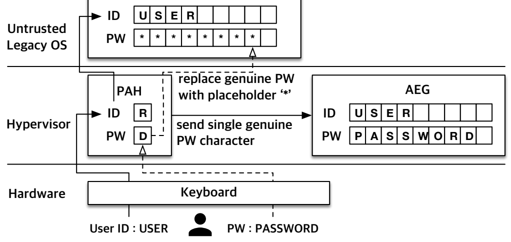


Figure 4: Port Access Handler.

Initial state	
Client	
Device (CD):	PCR17←The measurement of the hypervisor
CD:	has PCR17, locality
TAaaS:	has PCR17, locality, nonce, the same OTP with the user
Authentication evidence generation	
TAaaS→CD:	nonce
User→CD:	ID, PW, OTP
CD(AEG):	Extend(PCR21, ID || Hash(PW))
AEG:	Extend(PCR22, OTP)
AEG:	e1←Quote2(PCR17, PCR21, PCR22, nonce, locality)
Authentication evidence verification	
SP→TAaaS:	h1←Hash(ID || Hash(PW))
CD→TAaaS:	e1
TAaaS:	h2←Hash(OTP)
TAaaS:	e2←Hash(PCR17 || h1 || h2 || nonce || locality)
TAaaS:	if (verify(e1, e2, AIK_{pub})) then verification success

Figure 5: Simplified semantics for authentication evidence generation and verification.

5) The agent obtains the result of the TPM `Quote2` operation in response to `hc_req_evidence`.

After computing the authentication evidence, the agent forwards the evidence and the AIK certificate to TAaaS. Then, TAaaS performs the following verification process:

1) TAaaS already knows the valid PCR17 and locality because the hypervisor is affiliated with TAaaS.
2) The SP sends the correct hash digest of the user account information to TAaaS. Therefore, TAaaS knows the legitimate hash digest of the user account information (h1) corresponding with the PCR21 value.
3) Because it is the OTP service provider, TAaaS can generate the identical OTP to the user's one and compute the hashed OTP (h2) corresponding with the PCR22 value.
4) TAaaS recomputes a hash digest of the above values (e2).
5) Finally, TAaaS checks that the authentication evidence (e1) matches the recomputed digest (e2) by verifying the evidence with the public AIK, which is extracted from the AIK certificate.

TAaaS verifies the authentication evidence with *the hash value of the user account information*, not the actual one. Therefore, the actual user account information of the SP cannot be disclosed in the cloud.

D. Security Analysis

In this subsection, we perform the informal security analysis of the TAaaS model and show our work is resistant to the adversary models described in Section III-B.

Attacks in the client device. The adversaries try to acquire the authentication evidence to deceive TAaaS.

First, the adversaries may attempt to generate the authentication evidence. However, they cannot acquire any actual authentication data because they can capture only placeholders.

Second, the adversaries may attempt to intercept the evidence to impersonate a legitimate user. However, because TAaaS manages session IDs and nonces in pairs, the authentication evidence should be changed depending on the session. Therefore, the adversaries cannot reuse the authentication evidence in other sessions.

Last, the adversaries may attempt to neutralize the hypervisor-based trusted path by a hyperjacking attack. However, TAaaS can detect this attack because the measurement of the hypervisor is included in the evidence.

Attacks in the cloud. The adversaries might aim to obtain the user account information of the SP from TAaaS. Because TAaaS handles only the hash value of the account information, the adversaries cannot acquire any valid information about the user.

V. Evaluation

A. Implementation

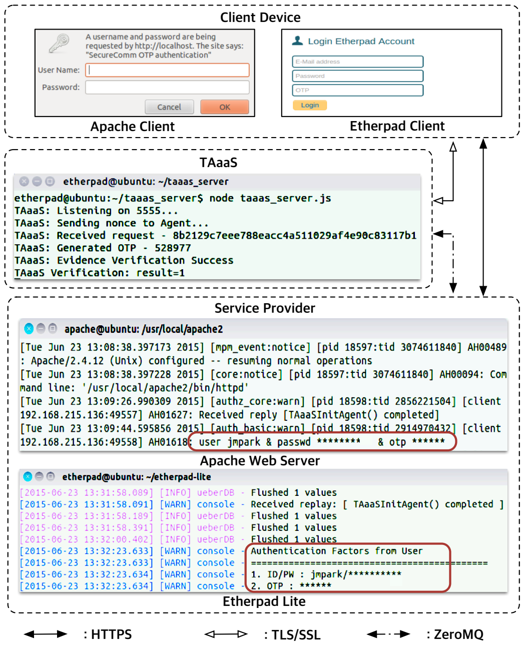
Client implementation. We implemented the hypervisor-based trusted path and the agent on a HP ProBook 6555b equipped with an AMD Turion P520 2.30GHz processor, and 4GB memory. The trusted path was implemented by leveraging XMHF [16] that supports a tiny hypervisor in a small TCB. The guest OS is the 32-bit version of Ubuntu 12.04.

XMHF provides development of custom hypervisor-based solutions, called hypapps. We implement a hypapp that consisted of a base module, a TPM, the HH, the PAH, and the AEG. The base module manages shared data with an OS, memory allocation, etc. The TPM component defines all the TPM operations in the hypervisor. The HH and the PAH are implemented by modifying two event handlers of the hypapp, which are responsible for processing the hypercall and handling the I/O port interceptions, respectively. The AEG is a function of the hypapp used to generate the authentication evidence. The agent is a Node.js-based application, and invokes the hypercalls via a library written in C.

TAaaS implementation. TAaaS is a network application using Node.js running on the Ubuntu 14.04 LTS machine. TAaaS acts as a TLS/SSL server to communicate with the agent in the client device and a ZeroMQ [17] server to interwork with the SP. We implemented the library, which generates OTP value and verifies the authentication evidence, in C. The Google Authenticator was used to generate the OTP value, so that users can use their own mobile devices as the OTP token by installing the Google Authenticator application.

B. Use Cases

We ported Apache HTTP Server and Etherpad to our TAaaS model. Figure 6 depicts the end-to-end implementation of two use cases. We modified the components for displaying the login page and processing the user authentication, to invoke the TAaaS APIs using ZeroMQ. Because



```
etherpad@ubuntu: ~/taaas_server
etherpad@ubuntu:~/taaas_server$ node taaas_server.js
TAaaS: Listening on 5555...
TAaaS: Sending nonce to Agent...
TAaaS: Received request - 8b2129c7eee788eacc4a511029af4e90c83117b1
TAaaS: Generated OTP - 528977
TAaaS: Evidence Verification Success
TAaaS Verification: result=1
```

```
apache@ubuntu: /usr/local/apache2
[Tue Jun 23 13:08:38.397173 2015] [mpm_event:notice] [pid 18597:tid 3074611840] AH00489
: Apache/2.4.12 (Unix) configured -- resuming normal operations
[Tue Jun 23 13:08:38.397228 2015] [core:notice] [pid 18597:tid 3074611840] AH00094: Com
mand line: '/usr/local/apache2/bin/httpd'
[Tue Jun 23 13:09:26.990309 2015] [authz_core:warn] [pid 18598:tid 2856221504] [client
192.168.215.136:49557] AH01627: Received reply [TAaaSInitAgent() completed]
[Tue Jun 23 13:09:44.595856 2015] [auth_basic:warn] [pid 18598:tid 2914970432] [client
192.168.215.136:49558] AH01618: user jmpark & passwd ********   & otp ******
```

```
etherpad@ubuntu: ~/etherpad-lite
[2015-06-23 13:31:58.089] [INFO] ueberDB - Flushed 1 values
[2015-06-23 13:31:58.091] [WARN] console - Received replay: [ TAaaSInitAgent() completed ]
[2015-06-23 13:31:58.189] [INFO] ueberDB - Flushed 1 values
[2015-06-23 13:31:58.391] [INFO] ueberDB - Flushed 1 values
[2015-06-23 13:32:00.402] [INFO] ueberDB - Flushed 1 values
[2015-06-23 13:32:23.633] [WARN] console - Authentication Factors from User
[2015-06-23 13:32:23.633] [WARN] console - =========================================
[2015-06-23 13:32:23.634] [WARN] console - 1. ID/PW : jmpark/**********
[2015-06-23 13:32:23.634] [WARN] console - 2. OTP : ******
```

Figure 6: TAaaS Use Cases.

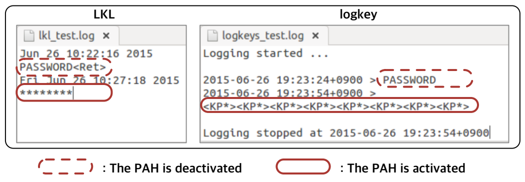
only placeholders are shown in the SP, the use cases decide the allowance of their services depending on the verification result from TAaaS. We explain how they work based on TAaaS in the following subsection.

Apache HTTP server. Apache HTTP Server (Apache) is the most representative and widely used web server. We modified three modules in Apache: `authz_core`, `auth_basic`, and `authn_file`. `authz_core` provides core authorization capabilities, and `TAaaSInitTP` is invoked by this module. `auth_basic` allows the use of HTTP Basic Authentication, and is modified to handle the OTP value along with the password via single "Password" field. `authn_file` provides user authentication, and is revised to invoke `TAaaSReqAuth` with the hash value of both ID and the password.

Etherpad. Etherpad is a web-based collaborative real-time editor and an example of Software as a Service. We install two plugins `ep_user_pad` and `ep_user_pad_frontend`, the user-management system for Etherpad. `ep_user_pad` provides the admin tool for `ep_user_pad_frontend`, the user-interface part. We modified `ep_user_pad_frontend` to invoke two cloud APIs and to add the input field for OTP value. Then, this revised module generates the hash value of both ID and the hashed password stored in the MySQL database, and calls `TAaaSReqAuth` to authenticate the user.

C. Formal Verification of Protocol

In order to verify the security of the TAaaS protocol, we leveraged ProVerif [18], a formal cryptographic protocol



```
Jun 26 10:22:16 2015
PASSWORD<Ret>
Fri Jun 26 10:27:18 2015
********
```

```
Logging started ...

2015-06-26 19:23:24+0900 > PASSWORD
2015-06-26 19:23:54+0900 >
<KP*><KP*><KP*><KP*><KP*><KP*><KP*><KP*>

Logging stopped at 2015-06-26 19:23:54+0900
```

Figure 7: Experiments against well-known keyloggers.

verification tool, and designed a communication environment using the cryptographic primitives and the entities in our model. The cryptographic primitives included the TPM `Extend` and `Quote2` operations. The entities consisted of the hypervisor, the SP, and TAaaS. We assumed that the hypervisor can obtain the input data securely because TAaaS can identify whether the hypervisor is compromised. Moreover, we considered the communication path between the hypervisor and the SP or TAaaS as a public channel, which can be attacked by adversaries.

Based on the assessment result, we confirmed that adversaries cannot acquire any authentication data. The ProVerif script file for TAaaS is available at [19].

D. Experiments on Existing Keyloggers

We performed an experiment to show that our hypervisor-based trusted path is robust against the well-known keyloggers: LKL version 0.1.1 and Logkeys version 0.1.1b. The LKL sniffs every byte that passes through the keyboard port via `inb(port)`. The Logkeys records keystrokes by reading events in `EV_KEY`, which describes state changes of keyboards. As shown in Figure 7, the password ('PASSWORD') is sniffed by the keyloggers when the PAH is deactivated. However, while the PAH is activated, only placeholders can be sniffed. Thus, the untrusted legacy OS cannot affects the trustworthiness of user inputs.

E. A Small TCB

A small TCB size is considered to be a paramount factor for evaluating the security system. Table I shows the lines of code (LoC) counted by CLOC [20] for the hypervisor-based trusted path on XMHF. We only add 1,599 LoC to the core implementation of XMHF, thus total LoC is 7,617. As compared with the LoC of other hypervisors, which aims to minimize the TCB, in [16], TAaaS is comprised of the smallest LOC.

Table I: Added LoC for the hypervisor-based trusted path.

	TPM	PAH	HH	AEG	Base module	Total
LoC	288	86	31	425	1,057	1,599

VI. Discussion

A. Limitations

Overhead for the hypervisor-based trusted path. Building the hypervisor-based trusted path requires additional overhead caused by the hypercalls and the TPM operations. We represented this overhead by measuring the time delay using CPU cycles during performance of the intended operation on a single character in the PAH, and by generating the authentication evidence in the AEG. We repeated the measurement 30 times, and calculated the average.

The average time taken in the PAH was approximately 3.19 ms. Except for the initial execution (1,004.37 ms), the average time taken by the AEG was about 549.10 ms. The initial overhead is caused by the TPM `LoadKey2` operation to load the wrapped AIK at boot time. However, this unwrapping is performed only once after booting up the client device.

We consider the use cases in Section V as examples. The lengths of the password ('PASSWORD') and OTP value are eight and six respectively, so that 44.66 ms is consumed by the PAH. As a result, the use cases needed roughly 593.76 ms to establish the hypervisor-based trusted path in the client device. Stuart K. Card et al. [21] defined the immediate response as less than one second, so that the overhead for the hypervisor-based trusted path might provide an acceptable user experience.

Phishing attack. The current TAaaS model does not consider phishing attacks intended to obtain authentication data with a fraudulent webpage. Because a phishing attack is a social engineering attack, it is very hard to counteract technically. However, an enhanced version of TAaaS could be a good alternative defense against phishing attacks. For example, the TAaaS architecture could be extended to provide a hot key that toggles the current input mode to the secure mode when the user presses it. In the secure mode, the phisher could obtain only placeholders instead of genuine authentication data.

B. Possible Application

Every SP can enhance the security of its user authentication by adopting TAaaS. For example, TAaaS can be adopted for MFA-based login systems like Google login with 2-step verification. TAaaS can be applied to cloud services for the government like AWS GovCloud (US) [22] to restrict access only to TAaaS-enabled client devices. Moreover, if the 2-factor authentication is a mandatory requirement of SPs like business banking [23], TAaaS can be deployed to provide their services with guaranteed trustworthiness.

VII. Related Work

A. Trusted Path

Researchers have proposed two other trusted path proposals: *local domain* and *remote domain*.

Local domain. The trusted path for local domain is established *inside the client device*. TrustLogin [24] leverages System Management Mode to guarantee the trusted path for protecting login credentials even when the OS is compromised. This work only focuses on the security inside the client device because it reveals the genuine password when the login packets arrive at the network card. In contrast, the TAaaS architecture ensures a trusted path between the user inputs and TAaaS. Zongwei Zhou et al. [12] proposed a hypervisor-based design for enabling a general-purpose trusted path. This work suggests concrete changes in the I/O architecture that would make future trusted path system design simpler. The hypervisor for TAaaS depends on such a trusted path to the keyboard, and could gain benefits from several techniques they propose. SecureSwitch [25] is a hardware assisted mechanism for secure instantiation and management of a trusted execution environment. It allows users to switch between a trusted OS and an untrusted OS via an enforced trusted path. DriverGuard [26] is a trusted hypervisor based protection mechanism for confidentiality of a driver's I/O flow against attacks from a malicious kernel.

Remote domain. The trusted path for the remote domain is established *between the client device and the remote server like TAaaS*. The Uni-directional Trusted Path system [27] is a secure transaction confirmation architecture via an isolated kernel that manages user-centric I/O devices. It enables a remote SP to gain assurance that client transactions are initiated by the action of humans and not by malware. TrustUI [28] provides a trusted path for mobile devices that offers secure interaction between mobile users and services based on the ARM TrustZone technology. Trusted Input Proxy [29] runs in a virtual machine and mediates I/O operations between physical device drivers and a guest OS to provide a secure network connection.

B. AaaS

AaaS is categorized into *Delegation* and *SSO*.

Delegation. Delegation is used to offload the verification of authentication data to AaaS by the SPs. Kemal Bicakci et al. [9] proposed that Mobile-ID server acts as the AaaS provider to authenticate secure element-equipped mobile devices. Rasib Khan et al. [10] proposed that an ATM offloads the verification of a user's transaction request to a cloud-based service using a one-time obfuscated PIN generated from the user's PIN and the SEPIA server.

SSO. SSO is used to leverage the authentication token generated by the AaaS to access the SPs. Rohitash Kumar Banyal et al. [7] proposed an MFA framework for cloud computing where cloud services and resources are classified into three levels, and different authentication factors are assigned to each level. Courtney Powell et al. [8] presented an inter-cloud SSO authentication architecture by which a user could manipulate the resources of inter-cloud systems that provide different Web APIs after being authenticated with the authentication portal. MFAaaS [11] is a cloud-based MFA architecture that leverages Identity Federation and SSO technologies to provide a modular integration of various authentication factors.

Unfortunately, the aforementioned works do not include consideration of security threats to the user inputs such as keyloggers or a compromised OS. Moreover, privacy problems caused by the risk of insider attackers of the cloud service provider could be raised if user account information is stored in, or is transferred to, the cloud.

VIII. Conclusions

In this paper, we propose TAaaS, a novel service model that offers a trusted path-based MFA service to SPs. This model leverages a tiny hypervisor-based trusted path to ensure the trustworthiness of user inputs. The trusted path is provided by revealing placeholders instead of genuine authentication data for secrecy, and verifying the authentication evidence for integrity. This approach also addresses privacy concerns in cloud by storing only the hash value of the user account information to TAaaS. We implemented two end-to-end prototypes to show feasibility, and performed experiments using well-known keyloggers to show the robustness of TAaaS. We verified the security of the TAaaS protocol using ProVerif, and showed the security analysis. We measured the overhead caused by the client device of the TAaaS architecture, and it appears endurable and acceptable with respect to the user experience.

As future work, we intend to extend the range of supported devices to various embedded devices, such as smartphone or in-home devices, by applying the ARM TrustZone-based trusted path as a decent alternative to the hypervisor.

References

[1] C. Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing v3. 0." *Cloud Security Alliance*, 2011.

[2] "StormPath," https://stormpath.com/product/login/.

[3] "SafeNet Authentication Service," http://www.safenet-inc.com/multi-factor-authentication/authentication-as-a-service/.

[4] "Symantec Validation and ID Protection Service (VIP)," http://www.symantec.com/vip-authentication-service/.

[5] Z. Xiao and Y. Xiao, "Security and Privacy in Cloud Computing," in *Communication Surveys and Tutorials*, 2013.

[6] N. Pathak, A. Pawar, and B. Patil, "A Survey on Keylogger: A Malicious Attack," in *International Jourcal of Advanced Research in Computer Engineering and Technology*, 2015.

[7] R. K. Banyal, P. Jain, and V. K. Jain, "Multi-factor Authentication Framework for Cloud Computing," in *Computational Intelligence, Modelling and Simulation (CIMSim), 2013 Fifth International Conference on*. IEEE, 2013.

[8] C. Powell, T. Aizawa, and M. Munetomo, "Design of an SSO authentication infrastructure for heterogeneous inter-cloud environments," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. IEEE, 2014.

[9] K. Bicakci, D. Unal, N. Ascioglu, and O. Adalier, "Mobile Authentication Secure against Man-in-the-Middle Attacks," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*.

[10] R. Khan, R. Hasan, and J. Xu, "SEPIA: Secure-PIN-Authentication-as-a-Service for ATM using Mobile and Wearable Devices," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*.

[11] A. U. S. Yogendra Shah, Vinod Choyi and L. Subramanian, "Multi-Factor Authentication as a Service," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*.

[12] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune, "Building Verifiable Trusted Path on Commodity x86 Computers," *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012.

[13] D. C. Latham, "Department of Defense trusted computer system evaluation criteria," *Department of Defense*, 1986.

[14] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *Security and Privacy (SP), 2010 IEEE Symposium on*.

[15] D. McKay, "A deep dive into hyperjacking," 2011.

[16] A. Vasudevan, S. Chaki, L. Jia, J. McCune, J. Newsome, and A. Datta, "Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework," in *Security and Privacy (SP), 2013 IEEE Symposium on*.

[17] "ZeroMQ," http://zeromq.org/.

[18] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *The Journal of Logic and Algebraic Programming*, 2008.

[19] "ProVerif script for TAaaS," http://goo.gl/w3OG9Z.

[20] "CLOC," http://cloc.sourceforge.net.

[21] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in *Proceedings of the SIGCHI Conference on Human factors in computing systems*. ACM, 1991.

[22] "AWS GovCloud (US) Region - Government Cloud Computing," http://aws.amaxon.com/govcloud-us/.

[23] Cryptomathic, "Two-Factor authentication for Banking - Building the Business Case."

[24] F. Zhang, K. Leach, H. Wang, and A. Stavrou, "TrustLogin: Securing Password-Login on Commodity Operating Systems," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*.

[25] K. Sun, J. Wang, F. Zhang, and A. Stavrou, "Secureswitch: Bios-assisted isolation and switch between trusted and untrusted commodity oses," in *NDSS'12*, 2012.

[26] Y. Cheng, X. Ding, and R. H. Deng, "DriverGuard: Virtualization-Based Fine-Grained Protection on I/O Flows," *ACM Trans. Inf. Syst. Secur.*, 2013.

[27] A. Filyanov, J. M. McCuney, A. R. Sadeghiz, and M. Winandy, "Uni-directional trusted path: Transaction confirmation on just one device," in *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, 2011.

[28] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li, "Building Trusted Path on Untrusted Device Drivers for Mobile Devices," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14. ACM, 2014.

[29] K. Borders and A. Prakash, "Securing Network Input via a Trusted Input Proxy," in *Proceedings of the 2Nd USENIX Workshop on Hot Topics in Security*, ser. HOTSEC'07, 2007.