# Too Much of a Good Thing: (In-)Security of Mandatory Security Software for Financial Services in South Korea

*Taisic Yun*[1,2] *Suhwan Jeong*[2] *Yonghwa Lee*[1]
*Seungjoo Kim*[3†] *Hyoungshick Kim*[4†] *Insu Yun*[2†] *Yongdae Kim*[2†]

[1] Theori Inc., [2] KAIST, [3] Korea University, [4] Sungkyunkwan University

## Abstract

Motivated by real-world hacking incidents exploiting Korea Security Applications (KSA) 2.0 from North Korea in 2023, we conducted a comprehensive security investigation into its vulnerabilities. For over a decade, KSA 2.0 has been mandated in South Korea for financial services, making it nearly ubiquitous on PCs nationwide. While designed to enhance security through measures such as secure communication, keylogger prevention, and antivirus protections, KSA 2.0 can bypass sandbox mechanisms, violating modern web security policies.

Our analysis uncovered critical flaws, including inconsistencies with web browser threat models, improper TLS usage, sandbox violations, and inadequate privacy safeguards. We identified 19 vulnerabilities that expose users to serious risks, such as keylogging, man-in-the-middle attacks, private key leakage, remote code execution, and device fingerprinting. These vulnerabilities were reported to government officials and vendors and have since been patched.

To understand the security implications of KSA 2.0, we conducted two user studies. First, our survey of 400 participants revealed widespread KSA 2.0 adoption, with 97% of banking service users having installed it, despite 59% not understanding its functions. Second, our desktop analysis of 48 users' systems found an average of 9 KSA installations per user, with many running outdated versions from 2022 or earlier. These findings suggest potential security concerns arising from the widespread deployment of KSA 2.0 in practice.

## 1 Introduction

Korea Security Applications (KSA) 2.0, a mandatory software suite widely implemented across South Korea, was introduced to secure financial and governmental online services. With over 207 million registered users as of 2022 [46], its widespread adoption is driven by historical legislation and stringent financial institution requirements. KSA 2.0 is intended to enhance the security of Internet banking by providing features that grant web pages privileged access to system resources that can be used for mandated security features beyond the capabilities of standard web browsers.

However, the effectiveness of KSA 2.0 in enhancing security is questionable. Recent vulnerabilities in KSA 2.0 have led to multiple real-world hacking incidents, particularly from North Korea [5–9, 58]. These attacks exploited flaws in KSA 2.0 programs to distribute malware and conduct supply chain attacks, compromising 61 institutions and over 10 million computers in 2023 [30]. KSA 2.0's widespread installation on most PCs, including those in government agencies, makes it a prime target and poses significant national security risks. This issue is not unique to South Korea; similar attempts to mandate potentially problematic software have been made globally, such as China's Green Dam Youth Escort in 2009 [47], Kazakhstan's root certificate installation attempt in 2019 [43], and Russia's Sovereign Internet Law [39]. These cases highlight the broader implications of government-mandated security solutions and emphasize the need to consider their potential risks carefully.

We first conduct a comprehensive security analysis of KSA 2.0 to evaluate its effectiveness and identify potential vulnerabilities. We examine the design and implementation of these applications, focusing on their interaction with system resources and web security models, using various methods such as black box testing, reverse engineering, and fuzzing with tools like AFL++ [23] and AFLnet [56]. Our findings reveal four design issues of KSA 2.0: inconsistencies between KSA 2.0 and web browser threat models, improper use of TLS protocols, violations of browser sandboxing, and enabling user tracking. We also discovered 19 vulnerabilities that demonstrate these design issues, leading to keylogging, man-in-the-middle attacks, private key leakage, remote code execution, and user identification. We reported all vulnerabilities to the respective developers, and they have been patched.

To better understand the security implications of KSA 2.0, we conducted two complementary user studies. First, we sur-

†: Corresponding authors

veyed 400 participants, with age and gender distributions matching South Korea's demographics. The survey revealed that 97% of users had installed KSA 2.0 for banking services, with 59% reporting they did not understand its functions. To validate these findings and assess actual installation patterns, we conducted a desktop analysis study with 48 additional participants. This analysis revealed an average of 9 KSA installations per user, with many systems running outdated versions from 2022 or earlier. Given that users perceive KSA 2.0 installation as a requirement for accessing essential services in Korea, this widespread deployment with outdated versions suggests that security vulnerabilities could significantly impact the national digital infrastructure.

Our contributions are summarized as follows:

- We conducted a comprehensive security analysis of KSA 2.0, identifying four major design flaws: inconsistencies with web browser threat models, improper TLS usage, browser sandbox violations, and enabling user tracking. We discovered 19 vulnerabilities that expose users to risks such as keylogging, man-in-the-middle attacks, private key leakage, remote code execution, and device fingerprinting.
- We conducted two user studies to understand the implications of widespread KSA deployment. Our survey of 400 participants, representative of South Korean Internet users, revealed that 97% had used KSA, while 59% were unaware of its functions. None of the participants identified the bank-mandated anomaly detection capabilities that could potentially enable user tracking. Our follow-up desktop analysis with 48 participants provided concrete evidence of security risks by revealing multiple outdated KSA versions per user, with many systems running versions from 2022 or earlier that lack critical security updates.
- We offer several actionable recommendations to enhance the security of banking services. Security vendors should focus on designing solutions that are well-integrated with browser architectures and follow the principle of least privilege. Governments play a significant role by establishing comprehensive regulatory frameworks that promote timely vulnerability and patch management and provide clear and detailed assessment guidelines.

Our work highlights the challenges of deploying non-standard security mechanisms, drawing on the case of South Korea's national-level experimentation with KSA.

# 2 Overview of KSA

In this section, we will cover the design goals of KSA and the evolution of its designs over time. A more detailed historical background is provided in the appendix (see Appendix A).

## 2.1 Design Goals

KSA is a suite of programs mandated for Korean citizens to securely use financial services and government services. It serves two main purposes: 1) supporting public-key based authentication on the web and 2) protecting sensitive data handled by web services, even if the user's device is compromised. KSA originated with the e-government initiative in South Korea in 1997. As part of this initiative, the government mandated regulated digital signatures for all Internet banking [42]. Early web browsers lacked the necessary security features, so banks provided their own through external plugins. In the 1990s, the Korean government developed a domestic encryption algorithm called SEED and mandated its use in public and financial sectors, aiming to protect domestic industries and promote cryptographic research [28]. Korean services also incorporated additional security features, such as firewalls, anti-keylogging, and anomaly detection, which were soon legally mandated for safer electronic finance [27].

## 2.2 Technical Issues

To achieve the aforementioned goals, KSA must access system resources. For instance, KSA requires access to device drivers or digital certificates stored in the file system to implement security features such as firewalls or certificate-based authentication. However, due to security concerns, modern browsers restrict such functionalities through sandboxing [18]. As a result, KSA *needs to violate the browser's security model and bypass the browser's sandbox*.

## 2.3 Early Design: KSA 1.0

In the initial version, KSA 1.0, ActiveX was used because Internet Explorer was the dominant browser at the time [4]. ActiveX, a software framework created by Microsoft, allowed native applications to interact through Internet Explorer, enabling web-based communication with native clients, which aligned perfectly with KSA's goals. As a result, KSA was widely adopted using this technology. However, ActiveX granted web-based access to various system resources, introducing significant security risks. Due to these concerns, Microsoft began deprecating ActiveX in 2015 [40].

## 2.4 Current Design: KSA 2.0

With the rise of alternative browsers like Chrome, Safari, and Opera, as well as the growing use of smartphones, the Korean government sought to amend the legal requirement for security programs reliant on ActiveX, which was tied to the Windows platform. However, service providers and security vendors continued using familiar designs despite these legal changes [62], likely due to the high costs and challenges of switching technologies, resulting in ongoing reliance on existing systems. After the deprecation of ActiveX, developers lost

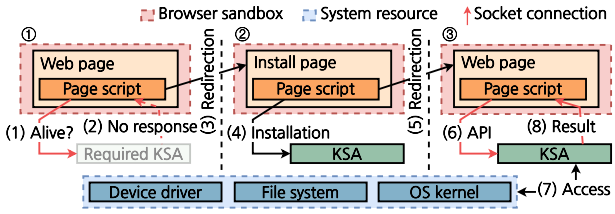


Figure 1: Workflow of KSA

a legitimate way to bypass the sandbox and began seeking new solutions. To address this, they discovered that web browsers can perform inter-process communication (IPC) through web protocols (e.g., HTTPS or WebSocket). Using this method, the web browser can interact with applications installed on the user's machine, bypassing the sandbox. While this approach allows developers to implement the desired features, it also introduces numerous security issues, which we will discuss in §5. In the following section, we will provide a more detailed explanation of how KSA 2.0 operates.

**Workflow.** Figure 1 illustrates the high-level workflow of KSA. ① When users access a specific website (e.g., a bank service web page), a script on the page attempts to connect to KSA. If the connection fails, the web page redirects users to the installation page. ② If KSA is not installed, users must download and install it from the installation page (see Figure 2), as they cannot use the service without it. In South Korea, many services mandate KSA installation even though legal requirements no longer enforce it. Users who refuse to install it cannot access services such as online banking services. ③ After installation, when users revisit the website, the script connects to KSA, which operates as a local server. The website can then use JavaScript to send requests to KSA for tasks requiring system resources (e.g., file read/write, system permission access). In this paper, we refer to these types of requests as KSA's API.

**Persistence and exposure.** During the installation process, KSA registers itself as a program that automatically starts up after a system reboot. Consequently, they continuously run in the background, maintaining open ports and actively waiting for connections from web pages requesting KSA services. This behavior keeps KSA programs exposed to not only legitimate web pages but also potentially malicious entities, increasing the attack surface and the risk of exploitation.

## 2.5 Current Status of KSA 2.0

**Ecosystem.** Third-party vendors develop KSA to meet the specific security requirements of financial companies and public service providers, such as top-tier banks, card and insurance companies, and some government services. The service providers then distribute these KSA solutions to users through their websites, mandating their use based on individual company policies.

**Popularity.** A recent technical report from the Bank of Korea reveals that 207.04 million users were registered for Internet banking in South Korea in 2022 [46]. To understand the requirements for KSA, we investigated all 17 top-tier banks in South Korea. Our findings indicate that 13 out of 17 banks state that installation is mandatory. However, upon further verification, we discovered that for 2 of these banks, users could still log in to the bank website without installation. Among the remaining 4 banks, 3 did not explicitly state that installation was mandatory but strongly recommended it, while 1 requested installation without explanation. To assess the prevalence of KSA usage among Internet users, we conducted an online survey with 400 representative participants. The survey results confirm the extensive and mandatory reach of KSA, with the vast majority of the population having installed it to access online banking services. The detailed study methodology and results are described in §6.

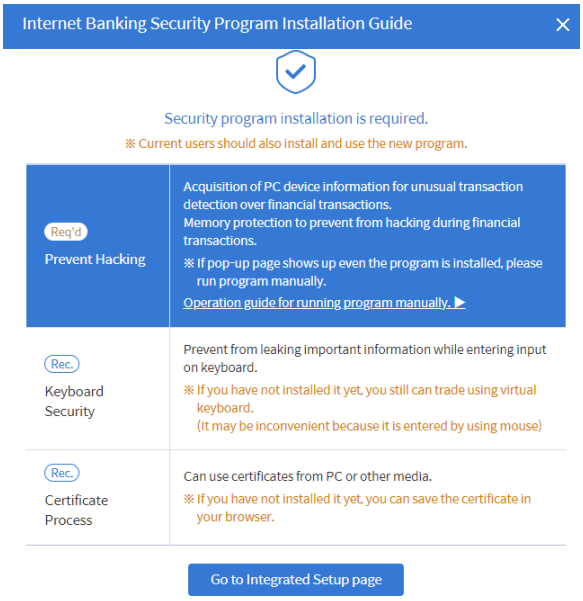


Figure 2: Example of an installation web page requiring users to install KSA programs

## 3 Threat Models based on Access Control

Before discussing the security analysis of KSA, we propose four threat models. To attack KSA, an adversary must be able to freely call KSA's APIs. The feasibility of these attacks varies depending on KSA's access control implementation. We do not include an attacker who can bypass all browser security techniques (i.e., a full chain exploit) in our models, as it is realistically challenging to circumvent all browser security measures.

## 3.1 Access Control

Figure 3 shows KSA's access control and the corresponding threat models. KSA's access can be controlled by three

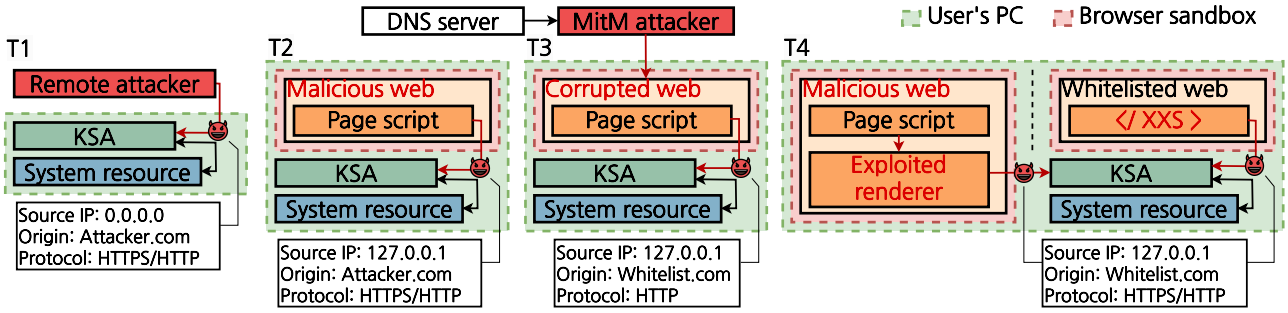


Figure 3: Threat models for KSA. Depending on KSA's access control, it is vulnerable to four types of attackers: A remote attacker (T1), malicious web (T2), Man-in-the-Middle (T3), an origin spoofer (T4).

factors: bind address, supporting protocols, and origin verification. The bind address determines which IP addresses can access KSA. If it is set to the wildcard address (i.e., 0.0.0.0), KSA can be accessed from anywhere on the Internet. Moreover, if KSA accepts insecure protocols (e.g., HTTP), it becomes vulnerable to man-in-the-middle attacks. Finally, the origin verification of KSA refers to the mechanism that checks the origin of a request using the HTTP Origin header [45]. This ensures that KSA can only be accessed by specified websites. These are existing mechanisms to limit attackers, yet many current KSA implementations lack even such basic access controls.

## 3.2 Threat Models

In this subsection, we discuss threat models corresponding to each access control.

**Remote attacker (T1).** The first threat model is named *Remote attacker*. This model is applicable when KSA allows access from the public Internet if KSA binds its address to the wildcard address (i.e., 0.0.0.0). In this case, an adversary can access KSA from anywhere on the Internet to exploit it, making it the most dangerous threat model.

**Malicious website (T2).** The second threat model is named *Malicious website*, which refers to an attacker capable of executing arbitrary JavaScript in a browser. This is the basic model for general browser security [13]. This threat model is useful if KSA does not check the origin of the API. In this case, the adversary can use JavaScript to call arbitrary APIs in KSA, exploiting its vulnerabilities.

**Man-in-the-Middle (T3).** The third threat model is named *Man-in-the-Middle*. This model signifies an attacker who can manipulate network packets (e.g., on public Wi-Fi). Considering the low adoption rate (27.4% [11]) of HTTP Strict Transport Security (HSTS), we assume that an attacker can make the victim access an HTTP page via the SSL stripping attack [38]. In this scenario, the attacker can execute arbitrary JavaScript, similar to **T1**, and can manipulate the HTTP Origin header, nullifying the origin check of KSA. Notably, in browsers, due to Mixed Content Blocking [21, 44], it is not possible to send HTTP requests in HTTPS domains, so we assume that the attacker can manipulate the network traffic.

**Origin spoofer (T4).** The final threat model is named *Origin spoofer*, which refers to an attacker who can bypass the origin check of KSA. Such attackers typically assume that the renderer is compromised or that there is a Cross-Site Scripting (XSS) vulnerability in whitelisted websites. As mentioned in previous research [29], renderer exploits are more common than sandbox escape (196 vs. 5 in 2022), making this more realistic than those allowing full chain exploits.

## 3.3 Comparison with Browser Extensions

KSA has a different security model compared to typical browser extensions [60]. Unlike extensions, KSA is not part of the browser's ecosystem and does not benefit from its protections. While the browser manages the origin check of extensions using its sandbox, KSA's origin check relies on the HTTP Origin, which is not protected by the browser's sandbox. Consequently, if the renderer is compromised (T4), KSA cannot securely validate the origin of the request, making it more vulnerable than browser extensions.

# 4 Analysis

## 4.1 Dataset

Table 1 shows mandatory KSA programs from top-tier banks in South Korea, which we used for our analysis. In South Korea, KSA can be divided into mandatory and optional programs. Users must install the mandatory KSA programs to access banking or public services, while optional ones are recommended by banking and public service websites for enhanced security. This paper focuses on the mandatory KSA programs from top-tier banks [12]; however, some of our target programs are also used in government services such as Korean Government Services (gov.kr) and Military Manpower Administration (mma.go.kr), though not all KSA programs deployed in government contexts were included in our analysis. Mandatory KSA programs are categorized into four main types:

- **Firewall.** Monitors packets and optionally blocks them if transmitted by unknown software.
- **Anti-keylogging.** Encrypts keyboard inputs from the device driver to the webpage to prevent keylogging.
- **Anomaly detection.** Collects various information (e.g., IP address or hardware information) from the user's computer to detect abnormal transactions.
- **Certificate management.** Manages certificates used for login or financial transactions.

## 4.2 Analysis Approach

To comprehensively analyze KSA programs, we conducted analysis by combining the following three methods.

**Blackbox testing.** The first method we used for analysis was black-box testing. This applies to all types of KSA programs regardless of their protection mechanisms (e.g., obfuscation or anti-reversing techniques). We analyzed the JavaScript of existing services and test scripts for developers, allowing us to understand how KSA programs operate at a high level and identify potential logical issues (e.g., keylogging) that could occur in KSA programs (§5.1).

**Reverse engineering.** Secondly, we utilized reverse engineering. Although more effort-intensive than black-box testing, reverse engineering allows us to uncover implementation details and hidden APIs within KSA programs. Through this, we discovered a vulnerability that leads to Remote Code Execution (RCE) by combining hidden features (§5.3). Many KSA programs have adopted various techniques to prevent reverse engineering, making analysis difficult. To circumvent these techniques, we used tools like unlicense [24] to bypass well-known obfuscations, including Themida [49]. However, in some cases, such as PRODUCT E, we failed to reverse engineer due to proprietary protections. In such cases, we used a debugger to inspect runtime memory and analyze behavior.

**Fuzzing.** Lastly, we applied fuzzing to identify memory errors using AFLnet [56], a network protocol fuzzer. Since AFLnet mutates data at the security protocol layer (e.g., SSL) rather than the application layer, we modified it to mutate data at the application layer for more effective fuzzing of KSA. For KSA with prolonged SSL connection processes, we patched the binaries to replace socket communication with standard input/output (I/O) operations, allowing us to use AFL++ [23], a general-purpose fuzzer, for more efficient fuzzing.

# 5 Security Problems in KSA

By systematically analyzing KSA, we identified four key design issues in KSA that can lead to significant security problems. These issues are categorized as follows:

1. **Inconsistencies in Threat Models between KSA and Web Browser**: KSA often places more trust in web pages than typical software, potentially enabling high-privileged attacks.
2. **Disregard for the TLS Security Models**: KSA implementations weaken the security of TLS, making it susceptible to Man-In-The-Middle (MITM) attacks.
3. **Violation of Browser Sandbox**: KSA conflicts with browser sandboxing, allowing attackers to bypass browser security mechanisms and compromise systems.
4. **Enabling User Tracking**: KSA often neglects user privacy, enabling various tracking and fingerprinting methods.

Unsurprisingly, these design issues do not align with widely accepted security practices and principles, particularly in web security. Therefore, robust implementation is essential; without it, serious consequences can arise. Furthermore, the design can be exploited even without implementation flaws. In the following sections, we detail the specific security problems and case studies resulting from these design issues. Additional issues are presented in Appendix D.

## 5.1 Inconsistencies in Threat Models between KSA and Web Browser

The first issue arises from inconsistencies between the threat model of KSA and that of web browsers. Typically, browsers do not trust web pages as they come from untrusted sources. Consequently, browsers implement various security policies to protect the system from web pages (e.g., Same Origin Policy [2] or sandbox [18]). However, while analyzing KSA, we observed that KSA tends to trust web pages more than general software. Specifically, it allows web pages to control high-privileged system actions to prevent malicious activities of general software. This opens up the possibility for attackers to execute high-privileged attacks that are impossible on the ordinary web (e.g., keylogging) or to neutralize protection mechanisms provided by KSA itself.

**Case study: Keylogging - PRODUCT D and PRODUCT E.** We discovered a vulnerability where a user's keyboard inputs can be stolen by exploiting the very anti-keylogging KSA (PRODUCT D or PRODUCT E) designed to protect them. To understand this issue, it is crucial to first examine how anti-keylogging KSA operates. It works by encrypting keyboard inputs through a device driver hook and transmitting them directly to the web page, where they are decrypted (Figure 4). While this encryption is meant to secure keystrokes, it ironically introduces the potential for keylogging attacks.

This design is particularly problematic because it assumes that an adversary can log keystrokes but cannot access the web page. However, keylogging typically requires high-level system access, meaning that an attacker would likely also have control over the browser, undermining the entire premise of the protection mechanism.

This flawed design can lead to even more serious issues. We

Table 1: KSA 2.0 software we analyzed, along with their access control mechanisms and corresponding threat models that make them vulnerable to attacks. The names of the software have been anonymized at the request of the Korean government agency.

| | Type | | | | Access Control | | | | | | Threat Model | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Bind Address | | | Protocol Sec. | | | | | | |
| Name | C | K | F | A | Localhost | Wildcard (+Filtering) | Wildcard | SSL | None | Origin Check | T1 | T2 | T3 | T4 |
| PRODUCT A | ✓ | | | | ✓ | | | ✓ | | | | ✓ | | ✓ |
| PRODUCT B | ✓ | | | | ✓ | | | ✓ | | ✓ | | | | ✓ |
| PRODUCT C | ✓ | | | | ✓ | | | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| PRODUCT D | | ✓ | | | ✓ | | | ✓ | | ✓ | | | | ✓ |
| PRODUCT E | | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | | ✓ |
| PRODUCT F | | ✓ | ✓ | ✓ | | ✓ | | ✓ | | | | ✓ | | ✓ |
| PRODUCT G | | | | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | | ✓ |

C - Certificate management, K - Anti-Keylogging, F - Firewall, A - Anomaly detection

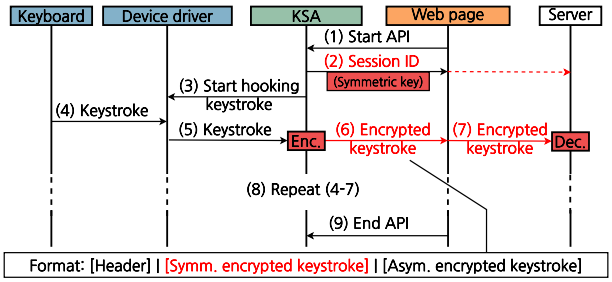


Figure 4: Workflow of PRODUCT E

found that anti-keylogging KSA can be abused for system-wide keylogging because it trusts the web page, allowing control over high-privileged system operations. This includes manipulating KSA to use symmetric encryption or even disabling encryption entirely. As a result, inputs can be captured not only from other web pages (SOP violation) but also from other programs (sandbox violation). This issue has been noted by other researchers [53], who confirmed the option to disable encryption, but it remains exploitable even in the latest patched version. Furthermore, we found that this issue affects other anti-keylogging KSA such as PRODUCT E. As shown in Figure 4, steps (2) and (6) involve sharing the symmetric key with the web page and encrypting the keystroke with this key, respectively, enabling keylogging. We agree that using symmetric encryption for keystrokes is nonsensical. During our disclosure process (see §5.5), vendors acknowledged this design vulnerability but stated they could not deprecate it due to reliance on this feature by existing services. As will be mentioned in §5.5, such a patch requires updates to both the server-side (e.g., JavaScript) and client-side components. As server-side patches require vendors to involve their customers, such as banking institutions, in the process, they are hesitant to proceed, risking a negative impression of the product.

**Case study: Disable security features - PRODUCT E and PRODUCT D.** Certain KSA (e.g., anti-keylogging) needs to provide APIs to enable or disable its functionalities. However, this renders these solutions ineffective in adversarial environments. This functionality is necessary because KSA is intended to be active only on specific websites, including banks. Otherwise, even after using the web service, all keystrokes by users would be encrypted by KSA, disrupting normal keyboard input. As the browser only knows which website is being accessed, KSA employs JavaScript on the pages to activate or deactivate its features (e.g., onfocusout event in DOM). However, this design allows attackers to call the said API, effectively neutralizing these solutions.

## 5.2 Disregard for the TLS Security Models

KSA supports TLS connections (e.g., HTTPS or WSS) but operates as a service on localhost, which is not well aligned with the typical model of TLS. To circumvent this, vendors have opted for a design that significantly weakens the security of TLS. That is, they install their own certificates for root CA and generate server certificates accordingly. We believe that this is due to the mixed content policy enforced by browsers in 2015, when KSA transitioned to a local service. At that moment, browsers do not allow mixed content (i.e., use of HTTP in HTTPS pages) in any domain, including localhost, and require the use of TLS. However, this is not the case anymore, as major browsers have eliminated the mixed content restrictions for localhost [10]. Thus, we believe that this is a legacy and outdated design.

Table 2: Root CA management for KSA

| Name | RootCA's certificate | KSA server's certificate | Delete RootCA's certificate after uninstall |
|---|---|---|---|
| PRODUCT A | G | G | |
| PRODUCT B | G | G | ✓ |
| PRODUCT C | F | F | |
| PRODUCT D | F | F | |
| PRODUCT E | F | G | |
| PRODUCT F | F | F | ✓ |
| PRODUCT G | G | G | |

F - "Fixed", G - "Generated at runtime"

**Case study: Untrustworthy CA - Every KSA.** To use TLS, vendors register their own CA, which significantly weak-

ens the security of TLS. As KSA operates in the localhost (127.0.0.1), it requires a valid certificate for the domain that will be resolved into the localhost. There are two ways for this: 1) using a certificate specifically for localhost (or 127.0.0.1), or 2) using a certificate for another domain and modifying the user's DNS settings (e.g., /etc/hosts) to point to localhost. Both methods are not feasible with conventional CAs; They do not issue a certificate for localhost, and it is practically impossible to provide individual certificates to each user. It is also infeasible to share the same certificate among users, as this would reveal the private key to all users.

As a result, KSA registers its own certificate as a Root CA during the installation process and generates the domain certificate accordingly. In this case, if the vendor of KSA is malicious or compromised, the TLS would be broken, enabling Man-In-The-Middle (MITM) attacks. Moreover, as shown in Table 2, all but two KSA implementations do not remove their Root CA certificates during uninstallation. This means that if the key is leaked, it complicates future patches and updates. Such a practice is considered extremely dangerous and should be avoided [16].

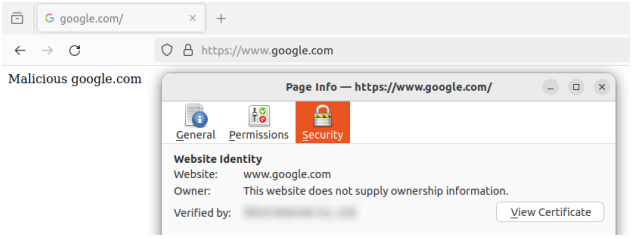


Figure 5: MitM attack using Google certificate issued by the PRODUCT E RootCA signed by the leaked private key

**Case study: CA private key leakage - PRODUCT E.** During our analysis, we confirmed a case of private key leakage, which we had been concerned about before. While investigating how KSA generates Root CA's certificate and server certificate, we discovered that one KSA generates a random server certificate at runtime based on a fixed Root CA's certificate (see Table 2). This implies that the private key of the Root CA should exist in a certain form during the installation process. Through reverse engineering, we identified that this private key is included in the binary with encryption and will be decrypted during the installation process. As a result, we could leak the private key of the Root CA and successfully sign the google.com domain with this key (see Figure 5). More seriously, the private key remains valid for any user's computer who has ever installed the KSA once. Even if the KSA is uninstalled, it continues to be effective unless the user directly modifies the OS certificate configuration.

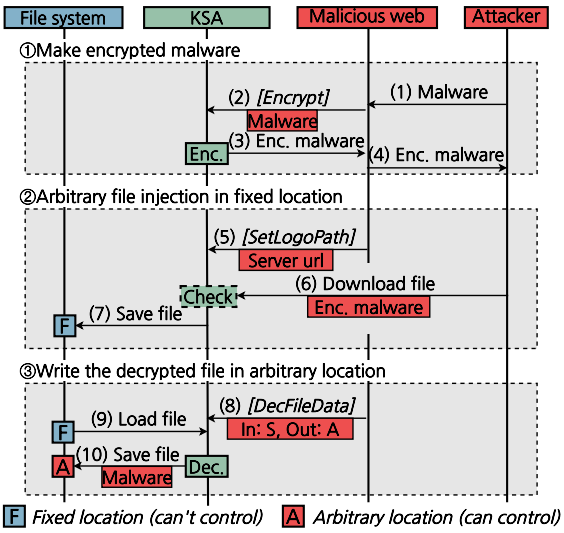


Figure 6: RCE attack workflow of PRODUCT B

## 5.3 Violation of Browser Sandbox

Browsers utilize sandboxes to prevent web pages from accessing system resources. Unfortunately, KSA inherently conflicts with the sandbox. KSA is located outside the sandbox because it is designed to implement functions that are not typically possible on the web. Consequently, through logical vulnerabilities or memory bugs within KSA, attackers can compromise systems without using browser vulnerabilities (e.g., renderer exploit or sandbox escape).

**Case study: Remote code execution - PRODUCT B.** We developed a remote code execution attack in PRODUCT B. This attack notably serves as a mere example among potential issues in KSA, implying the existence of other vulnerabilities that could be more severe.

In short, we found that PRODUCT B has two vulnerabilities that can be chained to achieve remote code execution (RCE). At first, we identified a vulnerability that allows us to move an encrypted file to an arbitrary location by decrypting it (i.e., arbitrary file move). However, within the sandbox environment, we cannot directly use this vulnerability to achieve RCE, as we can only access a sandbox directory with a random name, which cannot be predicted by an attacker. To resolve that, we use a second vulnerability to save an external file to an absolute path (i.e., arbitrary file download).

The workflow of this attack is illustrated in Figure 6. It is divided into three steps. ① The attacker uses KSA's API to encrypt the malware and receives the encrypted result. ② The attacker exploits a flawed error-handling mechanism within the SetLogoPath API to inject the encrypted malware to a specific fixed location on the user's system. KSA implements security measures to validate downloaded logos based on their digital signatures, but files that are 256 bytes or less can bypass this validation and are downloaded to a static location without being deleted. ③ The attacker uses the Dec-

FileData API provided by PRODUCT B to write the malware in a sensitive location. This API takes input and output files as parameters, decrypts the input file, and writes it to the output file. However, there is no validation process for the output file, allowing an attacker to write files to sensitive locations (e.g., Start Menu Program). By chaining these steps, we successfully achieved remote code execution by accessing a malicious web page.

**Case study: Memory Corruptions - PRODUCT F and PRODUCT G.** Currently, KSA is implemented in a memory-unsafe language (such as C or C++), making it vulnerable to memory corruption. To identify these issues, we used AFLnet [56], a fuzzer designed for network protocols. Memory corruption in KSA carries two significant implications due to its design. First, attackers can exploit these vulnerabilities directly without using browser vulnerabilities (e.g., renderer exploits and sandbox escapes). Second, attackers can abort KSA with a simple memory bug, disabling its security features. This is critical for KSA, which must provide security features precisely when needed. This effectively allows attackers to neutralize KSA, making further attacks easier.

Table 3: Overview of memory corruption bugs

| Name | OS | Type | Tool |
|---|---|---|---|
| PRODUCT F | Linux | Buffer overflow | AFL++ |
| PRODUCT G | Linux | Buffer overflow | AFLNET |
| | Linux | Buffer overflow | AFLNET |
| | Linux | Null Dereference | AFLNET |
| | Linux | Segmentation fault | AFLNET |

## 5.4 Enabling User Tracking

These days, browser privacy is increasingly recognized as a crucial issue, as evidenced by suggestions like using Incognito mode or removing third-party cookies. However, the current design of KSA lacks deliberate consideration for privacy, enabling various attacks like user tracking [57].

**Case study: Device fingerpriting - PRODUCT E.** In South Korea, KSA for anomaly detection is essentially a solution for fingerprint users. In particular, this KSA collects excessive PC information, raising significant privacy concerns. Here are the examples of information collected by this program:

- **Network info**: NAT IP, MAC address, IP, VPN IP
- **OS info**: Version, identification number, boot UUID
- **Hardware info**: Various hardware serial numbers
- **Others**: Firewall configuration, OS settings such as remote access permissions

Because of this, some researchers have mentioned that it is a nation-supported spyware [50]. Of course, users may be required to agree to the terms before installing such KSA. However, as we will discuss in §6, none of the participants who installed the required KSA for Internet banking were aware of its functionalities. These participants all utilized one of the top-tier banks, and most top-tier banks (13 out of 17) mandate at least one KSA with these capabilities.

More seriously, KSA allows not just banks but also attackers to obtain such information. Notably, KSA transmits this information to its server using public key encryption. Theoretically, this makes only the vendor of KSA, who has the private key, able to access this information. However, we have discovered a way to bypass this. It was possible through the use of KSA's test page. This page is for developers to test KSA. This test page is publicly accessible by anyone without any authentication. Therefore, an attacker can freely use it as a decryption oracle to decrypt encrypted data.

**Case study: User identification - PRODUCT A, PRODUCT B, and PRODUCT C.** Moreover, it is possible to deanonymize users using the certificate-based authentication KSA. This KSA contains an API to access the user's certificate. This certificate includes various information, such as certificate version, serial number, validity period, issuing authority, electronic signature algorithm, and user name. Among these, the user's legal name is particularly concerning as it can be used to identify the user, since it can be accessed in plaintex on the web. Moreover, the serial number can be used to track the user. These mechanisms conflict with the privacy and security model currently pursued by browsers. Recently, browsers have been putting a lot of effort into preventing querying users' identities to protect their privacy (e.g. incognito mode) [1]. The crucial point is that even when users utilize the incognito mode in browsers, KSA attackers can still identify the users' identities easily.

## 5.5 Responsible Disclosure

In early 2024, we conducted a closed meeting to inform government agencies and security vendors about the security issues and vulnerabilities we discovered. While we could have simply reported the vulnerabilities to security vendors through government-operated vulnerability disclosure programs, we felt the need to address fundamental root causes beyond mere code patches, as described in §5. Two government agencies — the Financial Security Institute (FSI) and the Korea Internet & Security Agency (KISA) — and one industry association — the Korea Information Security Industry Association (KISIA) — attended the meeting. In the meeting, we shared our opinions about the current state of KSA and the vulnerabilities we had discovered, along with the recommendations outlined in §7.2. Subsequently, we submitted detailed vulnerability reports to government agencies, who then forwarded them to the security vendors for necessary actions. For more details on the disclosure communication process, see Ethics considerations.

During the process of responsible disclosure, we encountered several challenges related to the distribution ecosystem and the responses from security vendors. These challenges, summarized below, highlighted significant problems in addressing fundamental vulnerabilities.

Table 4: Found vulnerabilities and current patch status. Some vulnerabilities have been fully patched, but in many cases, they were mitigated rather than completely patched. §5.5 explains the challenges of patching these vulnerabilities.

| Name | Design Issue | Vulnerability | Impact | Current Status | |
|---|---|---|---|---|---|
| | | | | Patched | Mitigated |
| PRODUCT A | §5.4 | Information leak of Accredited Certificate (e.g., user's name) | User identification | | ✓ |
| PRODUCT B | §5.4 | Information leak of Accredited Certificate (e.g., user's name) | User identification | | ✓ |
| | §5.3 | Lack of password validation | Encrypted private key leakage of Accredited Certificate | | ✓ |
| | §5.3 | Arbitrary file download & file move | Remote code execution | ✓ | |
| PRODUCT C | §5.4 | Information leak of Accredited Certificate (e.g., user's name) | User identification | | ✓ |
| PRODUCT D | §5.1 | Non-encryption option of the API | Keylogging | | ✓ |
| | §5.1 | Supporting deactivation API | Disable security feature (anti-keylogging) | | ✓ |
| | §5.1 | Incorrect understanding of access control | Origin check bypass | ✓ | |
| PRODUCT E | §5.1 | Using symmetric encryption | Keylogging | | ✓ |
| | §5.2 | Insecure certificate generation | Private key leakage of Root CA | ✓ | |
| | §5.1 | Supporting deactivation API | Disable security feature (anti-keylogging) | | ✓ |
| | §5.1 | Fail to manage sessions | Disable security feature (firewall) | | ✓ |
| | §5.1 | Incorrect threat model configuration | Bypass anomaly detection | | ✓ |
| | §5.4 | Insecure operation of the test server | Device fingerprinting | | ✓ |
| PRODUCT F | §5.3 | Memory corruption (1 found) | Disable security features | ✓ | |
| PRODUCT G | §5.3 | Memory corruption (4 found) | Disable security features | ✓ | |

**Complexity in applying patches.** The first major issue is that due to the unique ecosystem of KSA, it is challenging to distribute proper patches.

To patch KSA, its binaries, as well as the JavaScript code on each website of the service providers, should be updated. While typical JavaScript dependencies can be updated through established package management systems, KSA's JavaScript components are deeply integrated into service providers' existing web infrastructure and business logic. Therefore, updates cannot be treated as simple dependency updates.

Modifying JavaScript code requires significant changes to server-side code, and service providers would be hesitant to modify well-functioning production systems due to potential disruptions to their existing services. While conventional software updates often maintain backward compatibility during transition periods, the security-critical nature of KSA's vulnerabilities makes such gradual transitions problematic. According to the government agency, many vendors' clients were reluctant to abandon existing web functionalities, further complicating efforts to transition to updated systems. In addition, the lack of a unified patch deployment process exacerbates this challenge.

To address this, we reported the vulnerability to the security vendor using the existing channels and provided sufficient time for them to implement mitigations. However, these challenges demonstrate why it is not trivial to patch vulnerabilities that require changes to both KSA and JavaScript. For instance, to fix the keylogging vulnerability mentioned in §5.1, we need to remove the symmetric key feature of KSA. However, if we remove this feature only at the binary level, KSA may not work properly for services already using it. Due to this issue, security vendors initially mitigated the vulnerability by adding access control to restrict unauthorized access. While this approach mitigates immediate risks, stronger security models (e.g., origin spoofing attack [15]) may still allow attackers to exploit the same vulnerability. This underscores the need for a more comprehensive solution that effectively addresses the binary and JavaScript components.

Another critical aspect is that KSA is distributed not directly by the security vendor but by the service provider. This leads to significant version differences depending on the service provider, as shown in Table 5. Such an ecosystem complicates the timely deployment of patches, making it harder to ensure that all users are running the latest, most secure version. These issues are further amplified in certain government services with low security awareness. In extreme cases, these sites still use outdated technologies like ActiveX, prompting users to rely on outdated browsers such as Internet Explorer, which creates additional barriers to effective patching and increases overall security risks.

Table 5: Versions and release dates of PRODUCT E provided by financial institutions and public services

| Institution | Version | Release date |
|---|---|---|
| Insurance company A | 2019.2.20.1 | 2019-2-21 |
| Bank A | 2019.5.8.1 | 2019-5-8 |
| Credit card company A | 2020.6.21.1 | 2020-6-21 |
| Credit card company B | 2022.8.4.1 | 2022-8-4 |
| Bank B | 2023.1.9.2 | 2023-1-9 |
| Insurance company B | 2023.5.4.1 | 2023-5-4 |
| Public service A | 2023.6.30.9 | 2023-6-30 |
| Public service B | 2023.8.21.1 | 2023-8-21 |
| Public service C | 2023.8.22.9 | 2023-8-22 |

**Lack of incentive for security vendors to improve product safety.** Despite receiving detailed vulnerability reports and proof-of-concept (PoC) files, some security vendors exhibited insufficient or unsatisfactory responses in addressing the issues. For example, one security vendor responded to a reported vulnerability by adding an origin check instead of resolving the root cause despite there being a clear and feasible approach to patching the vulnerability effectively. More seriously, this mitigation was implemented incorrectly;

they blocked the localhost origin, used in our PoC, instead of whitelisting their client domains. This can prevent our PoC from working as it used the localhost origin, but it can not prevent an attacker from exploiting the vulnerability using their remote server. During our closed meeting with relevant agencies, we discussed the importance of proper origin checks and how to construct effective whitelists, and the agencies assured us that this feedback was communicated to the vendors. Despite this, the issue was not resolved correctly. We identified this issue and reported it through the established vulnerability reporting process, and the vendor eventually fixed the issue correctly.

**Current status.** With the cooperation of various institutions, we consistently provided feedback on vulnerabilities and patches to address the issues comprehensively. This has been done through the established vulnerability disclosure process. This collaboration led to the successful implementation of patches for all vulnerabilities that could be addressed immediately. However, due to the unique ecosystem of KSA, some tasks remain that require long-term efforts to address fundamental vulnerabilities. In these cases, security vendors have taken temporary measures by enhancing access control as an interim solution. This approach allows for immediate risk reduction while they plan and implement more comprehensive fixes over a longer period. Moreover, government agencies are attempting to remove legacy systems and security-risky features to address the issues identified in our analysis. For that, we agreed to impose an embargo until May 2025 to fully resolve the issues we identified. For the current patch status, see Table 4.

## 6 User Study

To understand the security implications of KSA in South Korea, we conducted two complementary studies. First, we conducted an online survey with 400 participants to assess the prevalence of KSA adoption and users' understanding of its security features. Second, we performed a desktop analysis study with 48 participants to precisely measure the number and versions of KSA programs installed on users' computers. Through these studies, we aim to quantify both the scope of vulnerabilities introduced by widespread KSA deployment and the potential risks stemming from gaps in user comprehension and practices.

### 6.1 Online Survey Methodology

**Research questions.** This user study aimed to answer the following research questions that complement our security vulnerability analysis of KSA:

**RQ1:** How prevalent is the installation and use of KSA among users?

**RQ2:** How well do users understand KSA's functionality and associated security risks?

RQ1 examines the extent of KSA 2.0 adoption among South Korean Internet users. Due to the lack of official statistics on KSA 2.0, our study provides an empirical estimate of its prevalence. In RQ2, we focus on identifying gaps between users' understanding of KSA's security features and their actual practices that could increase vulnerability risks. This involves assessing whether users recognize the potential security risks of KSA while uncovering common misconceptions about its security model that may leave them more susceptible to the identified vulnerabilities.

**Recruitment and demographics.** We recruited participants through `Embrain` (https://embrain.com/eng/), a professional survey company that maintains a diverse panel of respondents. Our study included 400 participants: 208 male (52%) and 192 female (48%). The age distribution spanned multiple decades: 20–29 years (21.5%, $n$=86), 30–39 years (26.75%, $n$=107), 40–49 years (26.5%, $n$=106), 50-59 years (16%, $n$=64), and 60 years or older (9.25%, $n$=37). To ensure the representativeness of our sample, we compared our demographic distributions against the national Internet user statistics provided by Statistics Korea (https://kostat.go.kr/anse/). Using Chi-square goodness of fit tests, we confirmed that both gender ($\chi^2 = 0.64$, $df = 1$, $p > 0.99$) and age ($\chi^2 = 2.31$, $df = 4$, $p > 0.99$) distributions were not significantly different from the population parameters.

**Survey design.** To ensure the clarity and accuracy of our survey questions, we conducted three pilot studies to refine the questionnaire. During our pilot test, we received feedback about the survey's duration (10 minutes) and insufficient detail in explaining the purpose of the Knowledge and Understanding section. We addressed this by providing a clearer and more comprehensive description of its objectives. The full survey questionnaire is available online (https://zenodo.org/records/14738628). The final 10-minute survey consisted of two main sections:

1. *Knowledge and Understanding:* This section assessed participants' knowledge and understanding of KSA. It included questions about the definition and scope of KSA, along with examples to provide clarity. Participants were asked to identify service types and statements related to KSA to gauge their level of familiarity with the subject.
2. *Awareness and Experiences:* This section focused on participants' hands-on experiences with KSA. It covered various aspects, including installation experiences, awareness of KSA functions and operations, and uninstallation methods employed by users.

### 6.2 Online Survey Results

Our survey results, summarized in Table 6, reveal key findings about KSA installation patterns and user awareness of its

Table 6: Survey results on user demographics and KSA practices[a]

| Question | Options | Response |
|---|---|---|
| **Gender** | Male | 52% (208) |
| | Female | 48% (192) |
| **Age** | 20s | 21.5% (86) |
| | 30s | 26.75% (107) |
| | 40s | 26.5% (106) |
| | 50s | 16% (64) |
| | 60+ | 9.25% (37) |
| **Experience with Banking & Government** | Banking & Government | 90.5% (362) |
| | Banking Only | 4% (16) |
| | Government Only | 3% (12) |
| | None | 2.5% (10) |
| **Bank KSA Installation Experience** | Installed | 97.35% (368) |
| | Not Installed | 0% (0) |
| | Unsure | 2.65% (10) |
| **Top-tier Bank Usage** | Yes | 100% (378) |
| | No | 0% (0) |
| **Uninstallation Methods** | Windows Uninstaller | 57% (228) |
| | Built-in Uninstaller | 8.75% (35) |
| | Third-party Tool | 6% (24) |
| | Manual Deletion | 4.5% (18) |
| | Never Uninstalled | 20.75% (83) |
| | Other | 0% (0) |
| | Unsure | 3% (12) |
| **Understanding of KSA Functions** | Yes | 40.75% (163) |
| | No | 59.25% (237) |
| **Knowledge of KSA Activation** | Always Active | 14.75% (59) |
| | Active on Access | 60.75% (243) |
| | Active on Specific Access | 20.75% (83) |
| | Unsure | 3.75% (15) |

[A] We first asked participants about their experiences with government and bank services. Specific questions regarding these services were directed only to those who had experience with each service type. As a result, the total number of responses to these specific questions may be less than the total number of survey participants (400).

security implications.

**RQ1 - Prevalence of KSA installation.** Our survey data demonstrates extensive KSA adoption across banking and government service users. Of our 400 participants, 90.5% had experience with both services, while 4% used only banking services and 3% used only government services. Among banking service users, KSA installation rates reached 97.35%, with the remaining 2.65% being unsure about their installation status. Notably, all 378 participants with banking experience reported using at least one top-tier bank that employed the target KSA analyzed in our work.

Our investigation of banking websites revealed that out of 17 top-tier banks, 11 required KSA installation for login, while six allowed access without it. However, 13 banks, including some that did not mandate KSA, used phrases like "*installation is required*" or "*need to install*" on their websites. Figure 2 shows an example of such an installation page. This practice, combined with the legacy of previous legal mandates, has contributed to KSA's widespread installation.

**RQ2 - Understanding of security implications.** Our analysis revealed significant gaps in users' awareness of KSA's security features and operation. Among participants, 59.25% reported not understanding KSA's functions. Regarding activation patterns, only 14.75% correctly identified that KSA programs run continuously as active services, while 60.75% believed they were active only during service access, and 20.75% thought they operated only during specific operations. The remaining 3.75% were unsure about activation patterns. Notably, no participants identified the anomaly detection capabilities that most banks require, a feature that accesses various information on users' PCs (see §5.4).

The survey also revealed concerning patterns in uninstallation practices. While 57% of participants reported using the Windows uninstaller and 8.75% used the built-in uninstaller, these methods fail to completely remove KSA components, particularly RootCA certificates. Only 10.5% of users employed methods capable of complete removal (6% using third-party tools and 4.5% performing manual deletion). Moreover, 20.75% had never attempted to uninstall KSA, leaving outdated versions on their systems without automatic updates (see §5.2). This combination of incomplete uninstallation methods and retained outdated versions leaves 86.5% of users exposed to potential security risks even after attempted uninstallation.

## 6.3 Desktop Analysis Study

Online surveys rely on self-reporting, which makes it difficult to verify the actual number of KSA programs installed on each user's PC. To address this issue, we conducted an additional user study. We installed a program called `HoaxEliminator` [61], designed to remove unnecessary Internet programs, including KSA, from participants' PCs to determine the number of KSA programs installed on each computer (multiple products, each uniquely tied to one installed version). Since the online survey platform we commissioned did not allow the installation of programs on users' PCs, we recruited additional participants who could voluntarily install the program through our institution's bulletin board and social networks. A total of 48 participants took part in this study: 34 male (70.83%) and 14 female (29.17%). The age distribution spanned multiple decades: 20–29 years (91.67%, $n$=44), 30–39 years (6.25%, $n$=3), 40–49 years (0%, $n$=0), 50-59 years (2.08%, $n$=1), and 60 years or older (0%, $n$=0). Each participant installed `HoaxEliminator` on their PC and confirmed the number of installed KSA programs. We instructed `HoaxEliminator` to output a list of the identified KSA programs. After the confirmation process, we assisted each user in successfully removing `HoaxEliminator`. The example response of the survey using `HoaxEliminator` is available in Appendix C.

Figure 7 shows that, except for two participants, all participants had KSA installed on their PCs, with one participant having as many as 24 KSA programs installed. The average

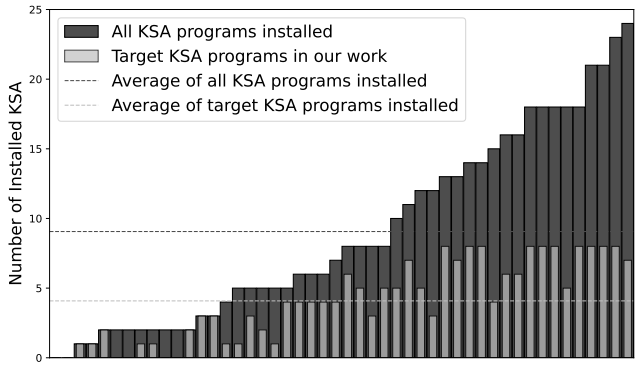


Figure 7: Number of installed KSA programs per user

number of total KSA installations was 9.06, with a standard deviation of 6.96, indicating a significant presence of KSA on users' computers. Focusing only on the specific KSA programs of interest in this paper, the average number of installations was 4.1, with a standard deviation of 2.92. Note that each installed KSA represents a distinct program, not different versions of the same program, as only one version per KSA can be installed on a user's computer.

When we checked the installed versions of each KSA, we found that many users have outdated versions of KSA installed. Out of 48 participants, 27 were using versions from 2022 or earlier. The oldest KSA found was installed on February 15, 2019. These findings suggest that old and insecure KSA programs persist on users' PCs.

## 6.4 Study Limitations

Our user studies provide essential insights into KSA usage and its security implications while also recognizing significant methodological limitations that require discussion.

**Online survey limitations.** Our survey faced two key methodological challenges. First, the survey design may have inadvertently influenced participant responses, particularly regarding mandatory installations and security perceptions. By asking about installation requirements, we may have drawn participants' attention to aspects they had not previously considered. To mitigate potential biases introduced by these perception-based responses, our primary analysis focuses on measurable factors such as installation status and usage patterns. While we collected data about users' views on mandatory installation requirements, we excluded these perception-based findings from our primary analysis. For transparency, these survey responses are available in Appendix B. Second, translating from Korean to English required careful consideration to preserve the original meaning. While we worked with bilingual experts to maintain accuracy and cultural relevance, some nuances may have been affected in translation.

**Desktop analysis study limitations.** The desktop analysis study provides concrete data on KSA program installations but is limited by participant demographics. The sample consisted of 48 users, predominantly male and aged in their 20s and 30s, recruited through institutional channels, which may not be representative of the broader Korean population. Additionally, requiring participants to install the `HoaxEliminator` tool might have introduced self-selection bias, favoring technically proficient individuals.

Despite these limitations, the combination of our large-scale online survey and focused desktop analysis provides complementary perspectives that strengthen our findings about KSA usage patterns and security implications in South Korea.

# 7 Discussion

## 7.1 Lessons Learned

**Dangers of web security solutions controlling client devices.** Our study demonstrates that KSA's design conflicts with fundamental web browser security principles. Specifically, KSA grants web pages control over high-privileged system actions, allowing them access to system resources that browsers typically restrict. The developers implemented this through web APIs that communicate with local services, bypassing browser security controls. This design not only contradicts browser security models but also creates an environment where security vendors may implement features that browsers explicitly prohibit for security reasons. This architectural approach introduces significant vulnerabilities. When web pages can execute high-privileged operations, attackers might exploit these privileges to perform actions typically prevented by browsers, such as keylogging or accessing sensitive files on client devices. Our security analysis demonstrates that such privileged access can create new attack vectors without requiring browser vulnerabilities. This challenge extends beyond KSA to other security solutions, particularly in remote management tools. For example, the 2021 Kaseya VSA incident [19] demonstrated how attackers could exploit a remote management tool's privileged access to deploy ransomware across thousands of businesses globally. Such incidents emphasize why security solutions should carefully balance functionality with the principle of least privilege, particularly when implementing web-based management interfaces that control client devices.

**Risks of solutions deviating from standards.** Web standards implemented by commercial web browsers undergo thorough review and verification by various experts and communities. Through open discussions, these standards address potential security issues before implementation and maintain systems for responding to discovered vulnerabilities. For instance, discussions about APIs for accessing host machine resources, such as WebHID and WebUSB, have led to careful consideration of security implications. The WebKit community has raised specific concerns about the WebUSB API's security

and privacy implications [63], prompting further expert discussions before full implementation. Our analysis suggests that deviating from these established standards may introduce security risks. For example, KSA's use of vendor-specific CAs could potentially weaken TLS security by enabling MITM attacks if a vendor's CA certificate is compromised or improperly managed on user systems. Given KSA's widespread deployment across South Korea, such vulnerabilities could potentially enable nationwide supply chain attacks. Furthermore, as we observed, the security patch process lacks standardized procedures, with each company implementing different levels of responses to security issues.

**Risks of mandating security solutions.** Our analysis of KSA revealed several vulnerabilities that could be exploited by abusing existing functionalities or leveraging well-known exploit methods. Despite its widespread use, the persistence of these issues in KSA may be attributed to its mandatory requirement, compounded by the absence of effective rollback and patching mechanisms. These findings suggest that mandating specific technologies should be accompanied by robust maintenance and update policies to address emerging security concerns. The potential risks of mandating specific security technologies are particularly evident when examining state-level security incidents. Several documented cases show North Korean actors exploiting vulnerabilities in KSA [5–9]. Similar concerns have emerged in other countries. In 2009, China withdrew its mandate for Green Dam Youth Escort software due to security flaws [47]. In 2019, Kazakhstan's attempt to enforce root certificate installation faced opposition from tech companies [43], while Russia's Sovereign Internet Law raised technical concerns [39]. These cases demonstrate how mandated software requirements necessitate careful consideration of security implications.

## 7.2 Recommendations

The issues discussed in §5 reveal that both implementation problems and design flaws can lead to significant security vulnerabilities. Based on the lessons in §7.1, it is crucial to fundamentally rethink the design of a secure ecosystem for online banking services. Achieving this goal requires the combined efforts and clearly defined roles of security vendors and the government.

**Recommendations for security vendors.** Based on our security analysis of KSA, we propose several recommendations for security vendors developing web-based security solutions. First, our findings in §3.1 demonstrate the need for robust access control mechanisms. While implementing proper origin validation and maintaining whitelists cannot prevent all attacks, these measures can increase the complexity required for successful exploits. Second, our analysis revealed how combining multiple privileged APIs enabled serious vulnerabilities like RCE attacks (§5.1). To mitigate such risks, vendors should implement privilege separation by restricting file system operations to specific directories and requiring explicit user consent for elevated privileges. This approach aligns with the principle of least privilege while maintaining necessary functionality. Third, our investigation of TLS security models (§5.2) showed how reliance on outdated mixed content policies can introduce vulnerabilities. Vendors should align their implementations with current browser security standards, particularly regarding localhost connections and content policies. For web-based authentication, standards like WebAuthn [3] offer secure alternatives that maintain compatibility with modern browsers while eliminating the need for custom security mechanisms.

**Recommendations for governments.** Our analysis of KSA highlights the importance of establishing comprehensive security frameworks for widely deployed security solutions. Based on the challenges observed in our vulnerability disclosure process (§5.5), governments should establish clear protocols for vulnerability reporting and remediation across vendors to address the inconsistent security patch responses we observed. The persistence of outdated versions and varying patch management practices revealed in our study indicates the need for structured maintenance policies. Our desktop analysis found that many users were running KSA versions from 2022 or earlier, highlighting systemic challenges in security update deployment. Governments could establish guidelines for security updates, version management, and regular security assessments. For example, frameworks like EU's GDPR [48] and CISA's VDP [20] demonstrate how structured security management approaches can be implemented while maintaining flexibility in technical implementation. Given that security solutions in critical sectors like banking and government services can significantly impact national digital infrastructure, our findings suggest that governments should establish regulatory frameworks that ensure vendors maintain robust vulnerability management processes and timely security updates. Such frameworks should include clear guidelines for security assessments and reporting mechanisms to verify compliance.

# 8 Related Work

**Security analysis for KSA.** There have been a few studies about KSA previously. Kim et al. [25] highlighted the vulnerabilities of KSA 1.0 to advanced malware and phishing attacks, while also mentioning usability issues. Despite the transition to KSA 2.0, the same issues mentioned in this study still persisted. Palant delved into KSA 2.0 and wrote several technical articles. He criticized South Korea's banking security with conventional features [50–55]. Our work differs from Palant's by systematically identifying threat models, uncovering a broader range of vulnerabilities, and highlighting design issues in KSA 2.0. We also emphasize issues not only in online banking services but also in the mandatory use of software and related problems. Additionally, we con-

duct a user study to address widespread use and awareness, providing recommendations for improved security.

**Security analysis for other security applications.** Besides KSA, numerous security applications are becoming integral to our daily lives. Consequently, much research has been devoted to evaluating the security of these applications. In the case of keyboard security, Yim et al. [34, 35] and Liang et al. [36] showed that commercial solutions cannot perfectly protect user input against the attacker. Chatzoglou et al. [17] and Haffejee et al. [25] proposed diverse techniques to bypass antivirus detection, and Min et al. [41] successfully achieved the local privilege escalation in various antivirus software via new attack vectors. However, none of these works dealt with applications mandated by law or those with design issues similar to KSA.

**Access control issues with untrusted sources.** Three recent studies have examined software vulnerabilities, focusing on access control issues and the risks associated with untrusted sources. Kim et al. (2023) [29] explored security weaknesses in browser extensions, revealing critical vulnerabilities in 40 extensions, including privilege escalation attacks and theft of sensitive information. Lin et al. (2024) [37] focused on Visual Studio Code (VS Code) extensions, identifying security issues arising from untrusted sources, such as code injection and file integrity attacks. Koishybayev et al. [31] analyzed the security of GitHub CI by examining approximately 447k GitHub workflows. They discovered that 99.8% of these workflows were overprivileged, and 23.7% of them allowed attackers to execute arbitrary code. These works highlight the crucial role of access control in addressing security issues in software handling untrusted sources.

## 9 Conclusion

While KSA 2.0, a suite of software mandated for online banking and public services in South Korea, aimed to bolster security, its flawed design and implementation highlight the risks of over-reliance on security solutions without rigorous testing. Motivated by real-world hacking incidents exploiting KSA 2.0 from North Korea in 2023, we investigated its major vulnerabilities. Our analysis uncovered four critical design issues: inconsistencies in threat models between KSA and web browsers, improper use of TLS, violation of browser sandboxing, and enabling user tracking. We examined KSA programs used by all top-tier banks in South Korea and identified 19 vulnerabilities posing significant security risks, such as keylogging, MITM attacks, private key leakage, remote code execution, and device fingerprinting.

The lessons from KSA 2.0 are crucial for building secure and trustworthy Internet services. Our findings suggest that extensive control over client devices, often advocated by policymakers and security vendors for secure services, should be approached with caution, as these solutions may introduce new attack surfaces. Although the implementation vulnerabilities we reported have been fixed, the current KSA 2.0 design still violates web security principles, potentially compromising systems.

As Ross Anderson aptly notes [27], "*Proprietary security software offers at best a modest improvement and certainly cannot provide the Holy Grail of a trustworthy user platform. We have recommended that banks should minimize the use of external plugins.*" Our empirical analysis of KSA 2.0 validates this observation through concrete evidence, demonstrating that mandated technologies, though initially beneficial, can eventually become the Achilles' heel against rapidly evolving attackers, leaving systems vulnerable to unforeseen threats.

## Acknowledgments

We sincerely thank the reviewers and shepherd for their valuable feedback. We are also deeply grateful to the late Ross Anderson, whose work on Korean banking security and practical guidelines helped shape the security community's understanding of this critical area. His legacy continues to inspire us. We would also like to thank Wladimir Palant for motivating us to explore the security issues in KSA. This work was supported by the Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00400302, RS-2024-00438686, and RS-2022-II221199).

## Ethics considerations

We ensure ethical conduct throughout our study, adhering to established protocols for vulnerability disclosure and user research.

**Responsible Disclosure.** To ensure the ethical handling of vulnerabilities discovered during our technical analysis, we adhere to a structured and detailed responsible disclosure process. This process involved multiple stages of communication and verification between our team, a government agency, and the vendors, ensuring that vulnerabilities were responsibly addressed without exposing users to undue risks.

1. *Initial Reporting:* This section assessed participants' knowledge and understanding of KSA. It included questions about the definition and scope of KSA, along with examples to provide clarity. Participants were asked to identify service types and statements related to KSA to gauge their level of familiarity with the subject.
2. *Vendor Communication:* The government agency forwarded our report to the respective vendors. Based on the provided details, vendors reviewed the vulnerabilities and implemented patches to address them. After completing the initial patching, vendors reported their patching

progress and implementation details back to the government agency.

3. *Cross-Verification:* The government agency shared the patched software and information about the applied fixes with us. Our team then conducted cross-checks to verify the efficacy of the implemented patches. This included testing for remaining vulnerabilities, evaluating the correctness of the fixes, and ensuring adherence to the recommended patching methods.
4. *Follow-Up and Additional Guidance:* During the cross-verification process, we identified cases where additional patches were required or where initial patches were incorrectly implemented. For instance, some fixes failed to address the root cause of the vulnerability or introduced secondary issues. In these instances, we provided detailed feedback and further recommendations to vendors, ensuring that all vulnerabilities were adequately addressed.
5. *Timeline and Progress:* The entire process was coordinated with a patch timeline agreed to extend until May 2025. Vendors typically implemented initial patches and provided reports within two months of disclosure. This timeline ensured sufficient time for thorough patching and verification while maintaining the security of affected systems.

By following this multi-stage process, we ensured that vulnerabilities were addressed comprehensively, minimizing potential risks to users and systems. This detailed approach not only prioritized ethical standards but also helped foster trust and accountability among all stakeholders involved.

**Online Survey.** Our user study fully adhered to ethical standards, and we obtained Institutional Review Board (IRB) approval. Participants were informed about the survey's purpose, content, and procedures and were made aware that they could withdraw from the study at any time without any consequences. Participation was voluntary and limited to adults, and anonymity was strictly maintained by collecting only non-identifiable information, such as gender and age. For surveys requiring program installation, we provided participants with clear instructions on installation, program functionality, and uninstallation. Responses were limited to screenshots of final outputs to minimize data collection risks. Data collected during the study was securely stored and accessible only to authorized researchers to ensure participant privacy and data integrity.

**Desktop Analysis Study.** An additional user study was conducted to collect information about KSA 2.0 programs actually installed by users, providing insights into their prevalence and usage patterns in real-world environments. This study adhered to the same ethical standards as the online survey, including IRB approval, participant anonymization, voluntary participation, and secure data handling protocols.

**Embargo and Research Integrity.** To protect the confidentiality of identified vulnerabilities, the findings remain under embargo until vendors have fully addressed the reported issues. The embargo period is set to extend until May 2025, ensuring sufficient time for patch implementation. The public release of the USENIX paper will occur after this embargo period, eliminating the risk of prematurely exposing vulnerabilities to potential attackers. This timeline aligns with established ethical research practices and prioritizes user safety. By adhering to these principles, we ensured ethical integrity in both the technical analysis and user research components of our study.

## Open science

As outlined in the §5.5, our study adheres to principles of responsible disclosure while balancing open science commitments with security considerations. We are bound by an embargo, coordinated through government agencies, that restricts the release of high-level details about identified vulnerabilities until May 2025. However, as the public release of the USENIX paper is scheduled for after this embargo period, the paper's publication timeline is unaffected by the embargo.

Even after the embargo period, government agencies have stipulated that certain sensitive details, such as proof-of-concept (PoC) exploits, detailed vulnerability analyses, and product identifiers, cannot be disclosed due to ongoing security threats, including state-sponsored attacks from North Korea. Despite these restrictions, our research findings have been approved for inclusion in the paper, provided that all sensitive details are omitted.

While these constraints limit the granularity of our public disclosures, we believe this approach prioritizes user safety and aligns with ethical standards, enabling us to share meaningful insights without compromising security.

In addition to the above commitments, and in response to feedback from our shepherd, we have decided to include both the Korean and English versions of our user survey as part of our artifact submission. The inclusion of the Korean survey addresses the challenge of obtaining user studies from Asia and offers a valuable resource for future research efforts, particularly in contexts where translating surveys into the venue's language is a significant hurdle. This contribution aligns with our open-science principles and supports the broader community by fostering accessibility and reproducibility in cross-cultural research.

## References


[1] Privacy on the web. https://developer.mozilla.org/en-US/docs/Web/Privacy.

[2] Same-origin policy. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[3] W3c - web authentication: An api for accessing public key credentials, April. https://www.w3.org/TR/webauthn/.

[4] For world's most wired country, breaking internet monopoly is hard. *The Korea Times*, april 2012. https://www.koreatimes.co.kr/www/news/biz/2012/04/123_109059.html.

[5] Dream security 'security certification s/w (magicline)' vulnerability fix recommendation. *National Cyber Security Center of the Republic of Korea*, June 28 2023. https://www.ncsc.go.kr:4018/main/cop/bbs/selectBoardArticle.do?bbsId=SecurityAdvice_main&nttId=54336&pageIndex=2&searchCnd2=.

[6] Initech 'inisafe' latest security update recommendation. *National Cyber Security Center of the Republic of Korea*, March 30 2023. https://www.ncsc.go.kr:4018/main/cop/bbs/selectBoardArticle.do?bbsId=SecurityAdvice_main&nttId=32172&pageIndex=2&searchCnd2=.

[7] North korea s/w supply chain attack-related joint cyber security recommendation. *National Cyber Security Center of the Republic of Korea*, November 23 2023. https://www.ncsc.go.kr:4018/main/cop/bbs/selectBoardArticle.do?bbsId=SecurityAdvice_main&nttId=93471&pageIndex=1&searchCnd2=.

[8] Rok-uk joint cyber security advisory (dprk s/w supply chain attacks). *National Cyber Security Center of the Republic of Korea*, November 23 2023. https://www.ncsc.go.kr:4018/main/cop/bbs/selectBoardArticle.do?bbsId=SecurityAdvice_main&nttId=93472&pageIndex=1&searchCnd2=.

[9] Security certification s/w old version security measures guide. *National Cyber Security Center of the Republic of Korea*, November 7 2023. https://www.ncsc.go.kr:4018/main/cop/bbs/selectBoardArticle.do?bbsId=SecurityAdvice_main&nttId=88018&pageIndex=1&searchCnd2=.

[10] Mixed content, 2024. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

[11] W3techs - usage statistics of http strict transport security for websites, January 2024. https://w3techs.com/technologies/details/ce-hsts.

[12] Bank of Korea. Types of financial institutions in south korea, 2023. https://www.bok.or.kr/portal/main/contents.do?menuNo=200580.

[13] Adam Barth, Collin Jackson, Charles Reis, and Google Chrome Team. The security architecture of the chromium browser. In *Technical report*. Stanford University, 2008.

[14] boYon Hwang. "cancerous regulations" still complicate online shopping. *Hankyoreh*, 2014. https://english.hani.co.kr/arti/english_edition/e_business/631104.html.

[15] Yinzhi Cao, Vaibhav Rastogi, Zhichun Li, Yan Chen, and Alexander Moshchuk. Redefining web browser principals with a configurable origin policy. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

[16] Robert N. Charette. Diginotar certificate authority breach crashes e-government in the netherlands. *IEEE Spectrum*, September 2011. https://spectrum.ieee.org/diginotar-certificate-authority-breach-crashes-egovernment-in-the-netherlands.

[17] Efstratios Chatzoglou, Georgios Karopoulos, Georgios Kambourakis, and Zisis Tsiatsikas. Bypassing antivirus detection: old-school malware, new tricks. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–10, 2023.

[18] Chrome Developers. Sandbox. https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md.

[19] Cybersecurity and Infrastructure Security Agency (CISA). Kaseya ransomware attack: Guidance for affected msps and their customers. https://www.cisa.gov/news-events/news/kaseya-ransomware-attack-guidance-affected-msps-and-their-customers, 2021.

[20] Cybersecurity and Infrastructure Security Agency (CISA). Vulnerability disclosure policy (vdp) platform. https://www.cisa.gov/resources-tools/services/vulnerability-disclosure-policy-vdp-platform, 2023.

[21] Chris Evans and Tom Sepez. Trying to end mixed scripting vulnerabilities. *google blog*, June 2011. https://security.googleblog.com/2011/06/trying-to-end-mixed-scripting.html.

[22] Financial Services Commission. Regulation on supervision of electronic financial transactions. https://www.fsc.go.kr/eng/pr010101/21742.

[23] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. AFL++: Combining incremental steps of fuzzing research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, August 2020.

[24] Erwan Grelet. unlicense. https://github.com/ergrelet/unlicense.

[25] Jameel Haffejee and Barry Irwin. Testing antivirus engines to determine their effectiveness as a security layer. In *2014 Information Security for South Africa*, pages 1–6. IEEE, 2014.

[26] Chico Harlan. Explorer for online shopping because of security law. *The Whashington Post*, November 2013. https://www.washingtonpost.com/world/asia_pacific/due-to-security-law-south-korea-is-stuck-with-internet-explorer-for-online-shopping/2013/11/03/ffd2528a-3eff-11e3-b028-de922d7a3f47_story.html.

[27] Hyoungshick Kim, Junho Huh, and Ross Anderson. On the security of internet banking in south korea. CSRR 10-01, Computer Science Group, Oxford University Computing Laboratory, 2010.

[28] Young-Myung Kim. Korea to allow adoption of international standard encryption aes from 2026. *Boan News*, 2024. https://m.boannews.com/html/detail.html?tab_type=1&idx=132764.

[29] YoungMin Kim and Byoungyoung Lee. Extending a hand to attackers: browser privilege escalation attacks via extensions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 7055–7071, 2023.

[30] Donghwan Ko. North korea's lazarus hacked 61 s. korean agencies: police. *The Korea Times*, April 2023. https://www.koreatimes.co.kr/www/nation/2024/05/103_349281.html.

[31] Igibek Koishybayev, Aleksandr Nahapetyan, Raima Zachariah, Siddharth Muralee, Bradley Reaves, Alexandros Kapravelos, and Aravind Machiry. Characterizing the security of github ci workflows. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2747–2763, 2022.

[32] Korean Legislation Research Institute. Digital signature act. https://elaw.klri.re.kr/eng_service/lawView.do?hseq=48669&lang=ENG.

[33] Jaemyoung Lee. An internet banking hacking and robbery. *donga*, June 2005. https://www.donga.com/en/article/all/20050604/241683/1.

[34] Kyungroul Lee and Kangbin Yim. Keyboard security: A technological review. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 9–15. IEEE, 2011.

[35] Kyungroul Lee and Kangbin Yim. Vulnerability analysis and security assessment of secure keyboard software to prevent ps/2 interface keyboard sniffing. volume 23, page 3501. MDPI, 2023.

[36] Xinyue Liang and Jun Ma. A study on screen logging risks of secure keyboards of android financial apps. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 101–111. IEEE, 2022.

[37] Elizabeth Lin, Igibek Koishybayev, Trevor Dunlap, William Enck, and Alexandros Kapravelos. Untrustide: Exploiting weaknesses in vs code extensions. In *Symposium on Network and Distributed System Security (NDSS)*, 2024.

[38] Moxie Marlinspike. New tricks for defeating ssl in practice. *Black Hat DC*, 2, 2009.

[39] Sam Meredith. Russia's controversial 'sovereign internet' law goes into force. *CNBC*, 2019. https://www.cnbc.com/2019/11/01/russia-controversial-sovereign-internet-law-goes-into-force.html.

[40] Microsoft Edge Team. A break from the past, part 2: Saying goodbye to activex, vbscript, attachevent. May 2015. https://blogs.windows.com/msedgedev/2015/05/06/a-break-from-the-past-part-2-saying-goodbye-to-activex-vbscript-attachevent/.

[41] Byungho Min, Vijay Varadharajan, Udaya Tupakula, and Michael Hitchens. Antivirus security: naked during updates. *Software: Practice and Experience*, 44(10):1201–1222, 2014.

[42] Ministry of the Interior and Safety. All that digital gov. korea. Technical report, Government of South Korea, 2022. https://www.innovation.go.kr/en/bbs/resources/resDetail.do?bbsId=B0000062&nttId=9518.

[43] Elizabeth Montalbano. Kazakh government will intercept the nation's https traffic. *ITPro*, 2019. https://www.itpro.com/network-internet/34051/kazakh-government-will-intercept-the-nation-s-https-traffic.

[44] Mozilla. Mixed content blocking in firefox. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content/How_to_fix_website_with_mixed_content.

[45] Mozilla. Origin. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin.

[46] Kyungdon Nam. Internet banking customers up 8.5% in 2022. *The Korea Herald*, 2023. https://www.koreaherald.com/view.php?ud=20230322000824.

[47] BBC News. China defends 'green dam' software. *BBC News*, 2009. http://news.bbc.co.uk/2/hi/asia-pacific/8091044.stm.

[48] Council of the European Union. The general data protection regulation. https://www.consilium.europa.eu/en/policies/data-protection/data-protection-regulation/, 2024.

[49] Oreans Technologies. Themida overview. https://www.oreans.com/themida.php.

[50] Wladimir Palant. Ipinside: Korea's mandatory spyware, 2023. https://palant.info/2023/01/25/ipinside-koreas-mandatory-spyware/.

[51] Wladimir Palant. South korea's banking security: Intermediate conclusions, 2023. https://palant.info/2023/02/20/south-koreas-banking-security-intermediate-conclusions.

[52] Wladimir Palant. South korea's online security dead end, 2023. https://palant.info/2023/01/02/south-koreas-online-security-dead-end/.

[53] Wladimir Palant. Touchen nxkey: The keylogging anti-keylogger solution, 2023. https://palant.info/2023/01/09/touchen-nxkey-the-keylogging-anti-keylogger-solution/.

[54] Wladimir Palant. Veraport: Inside korea's dysfunctional application management, 2023. https://palant.info/2023/03/06/veraport-inside-koreas-dysfunctional-application-management/.

[55] Wladimir Palant. Weakening tls protection, south korean style, 2023. https://palant.info/2023/02/06/weakening-tls-protection-south-korean-style/.

[56] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. Aflnet: A greybox fuzzer for network protocols. 2020.

[57] Reason Cybersecurity Ltd. What is user tracking? https://cyberpedia.reasonlabs.com/EN/user%20tracking.html.

[58] Shreyas Reddy. North korean hackers target popular banking software in south korea: Nis. March 2023. https://www.nknews.org/2023/03/north-korean-hackers-target-popular-banking-software-in-south-korea-nis/.

[59] Simon Sharwood. South korea kills activex-based government digital certificate service. *The Register*, December 2020. https://www.theregister.com/2020/12/10/south_korea_activex_certs_dead/.

[60] Mike Ter Louw, JinSoon Lim, and Venkat N Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4:179–195, 2008.

[61] TEUS. Hoaxeliminator 7.25, 2024. https://teus.me/hoaxeliminator/HoaxEliminator7.25/.

[62] The Korea Times. Card firms to scrap activex for convenience. *The Korea Times*, 2024. https://www.koreatimes.co.kr/www/biz/2024/07/602_175761.html.

[63] Webkit. Webusb api. https://github.com/WebKit/standards-positions/issues/68, 2022.

## A History of KSA

Table 7: Comparison of KSA 1.0 and KSA 2.0

| | KSA 1.0 | KSA 2.0 |
|---|---|---|
| **Usage duration** | 2002 - 2015 | 2015 - Current |
| **Framework** | ActiveX | C / C++ |
| **Operation method** | ActiveX control | Local server (EXE) |
| **Mandating entity** | Law | Policies of each company |

This section explores the detailed background and evolution of KSA.

**Introduction of Accredited Certificate and ActiveX.** The use of KSA began with the Digital Signature Act [32]. In 1997, the government of South Korea promoted the construction of e-government [42]. The government enacted the Digital Signature Act in 1999 as part of these efforts. This law enabled electronic signatures, implemented with public-key certificates issued by nationally designated certificate authorities, to have the same legal effect as handwritten signatures. We refer to these kinds of certificates as Accredited Certificate[1] in South Korea. During the expansion of electronic signatures, Accredited Certificate became legally mandatory in extensive

[1] In more detail, as of December 10, 2020, the legal status of Accredited Certificate has been abolished and renamed to Joint Certificate with the revision of the Digital Signature Act. These changes have allowed for the issuance of various private certificates and enabled them to compete on an equal footing with Accredited Certificate in the marketplace. However, there is no difference regarding KSA, so in this paper, we refer to all as Accredited Certificate.

online domains such as Internet banking and online shopping malls in 2002 [42]. At that time, Accredited Certificate was stored on the hard disk, including their public key and encrypted private key with a password. Consequently, they required operations involving access to system resources (e.g., reading and writing files) through the web. This led the South Korean government to adopt a method based on ActiveX[2] technology, which is dependent on Internet Explorer (IE) [26], due to high reliance on Microsoft's browser. This technology enables web pages to access system resources, such as files and devices, which is not possible with ordinary web pages.

**KSA 1.0: ActiveX-based security mechanisms.** In 2005, an incident led to the mandatory use of security mechanisms [33]. In 2024, a hacker in South Korea installed a keylogger and obtained the password of Accredited Certificate. Subsequently, the hacker successfully embezzled funds from the bank, marking South Korea's first hacking incident in Internet banking. In response to this event, the government amended the Electronic Financial Supervisory Regulations in 2006 to enhance the security of Internet banking [22]. This regulation mandated the installation of security programs, including intrusion prevention systems and anti-keylogging software. Consequently, various security companies developed and deployed ActiveX-based solutions for Internet banking, which had already gained popularity for certificate-based authentication.

**Towards KSA 2.0.** South Korea experienced significant changes in its security mechanism in 2015. Firstly, foreigners faced issues accessing South Korea's Internet banking services as they were unable to obtain a Korean Accredited Certificate [14]. As a result, the Korean government decided to remove the mandatory use of Accredited Certificate in Internet banking. Additionally, Microsoft introduced Microsoft Edge as the successor to Internet Explorer, completely abandoning support for ActiveX [40]. Adapting to these changes, the South Korean government initiated the phase-out of KSA 1.0, based on ActiveX [59]. As part of these changes, the legal obligation to install security programs or use Accredited Certificate was removed. However, transforming a market that has been rigid under government-led security policies for a long time into one that allows for diversity is not easy. As a result, even though there is no longer a legal obligation, almost all financial companies and some public services have continued to compel users to install security solutions based on their policies, which have evolved into KSA 2.0.

## B Survey Results Reflecting Users' Opinions

Table 8 shows the results for all our survey questions.

**User perception and experience with KSA installation and usage.** We asked participants who used banking and government services about their perception of how often KSA installation is mandated. For banking users, 50.53% believed it was always mandatory, and 24.87% thought it was frequently required. For government services, 42.78% perceived it as always mandatory, and 28.34% believed it was frequently required. Only 1.06% and 1.07% of users, respectively, thought installation was never mandated. Additionally, when asked if they would have used KSA if it were not mandatory, 52.25% indicated they would not have.

Despite findings that KSA does not significantly enhance security and may pose threats, 5.75% of participants believe KSA is very helpful, and 38.25% believe it is somewhat helpful. Only 11.25% of participants responded that KSA is not helpful at all. However, given that only 40.75% of users reported understanding the functions provided by KSA, and no participant could accurately identify the specific functions offered in practice, it is questionable whether their responses regarding the security benefits of KSA are based on a clear understanding of the programs.

## C Example Response from the Survey using `HoaxEliminator`

Figure 8 is a sample response from the online survey participants. We blurred the product names to ensure confidentiality.

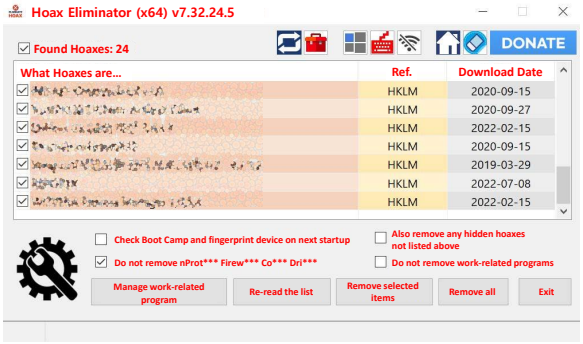


Figure 8: Example screenshot provided by respondents

## D Other Vulnerabilities in KSA

### D.1 Anomaly Detection Bypass

The anomaly detection KSA actively collects a wide array of information from internet banking users' PCs, including hardware specifications, real IP addresses, and proxy IPs, before transmitting this data to the bank's server. This collected data is crucial for identifying abnormal transactions, such as blocking transactions originating from high-risk countries or detecting unusual IP address changes. However, this design suffers from a fundamental flaw stemming from an oversight in the initial threat model setup. The vulnerability lies in the system's reliance on executable files installed on attackers' PCs to gather data on potentially illicit transactions. The data exchanged between KSA and the server is encrypted using

[2] ActiveX is a technology that allows web pages to execute native code on the client side.

Table 8: Entire Survey Results

| Question | Options | Response |
|---|---|---|
| **Gender** | Male | 52% (208) |
| | Female | 48% (192) |
| **Age** | 20s | 21.5% (86) |
| | 30s | 26.75% (107) |
| | 40s | 26.5% (106) |
| | 50s | 16% (64) |
| | 60+ | 9.25% (37) |
| **Experience with Banking & Government** | Banking & Government | 90.5% (362) |
| | Banking Only | 4% (16) |
| | Government Only | 3% (12) |
| | None | 2.5% (10) |
| **Bank KSA Installation Experience** | Installed | 97.35% (368) |
| | Not Installed | 0% (0) |
| | Unsure | 2.65% (10) |
| **Government KSA Installation Experience** | Installed | 94.39% (353) |
| | Not Installed | 0.27% (1) |
| | Unsure | 5.35% (20) |
| **Top-tier Bank Usage** | Yes | 100% (378) |
| | No | 0% (0) |
| **Bank KSA Enforcement Frequency** | Always | 50.53% (191) |
| | Frequently | 24.87% (94) |
| | Occasionally | 18.78% (71) |
| | Rarely | 4.76% (18) |
| | Never | 1.06% (4) |
| **Government KSA Enforcement Frequency** | Always | 42.78% (160) |
| | Frequently | 28.34% (106) |
| | Occasionally | 21.93% (82) |
| | Rarely | 5.88% (22) |
| | Never | 1.07% (4) |
| **Willingness to Use If Not Mandated** | Yes | 47.75% (191) |
| | No | 52.25% (209) |
| **Uninstallation Methods** | Windows Uninstaller | 57% (228) |
| | Built-in Uninstaller | 8.75% (35) |
| | Third-party Tool | 6% (24) |
| | Manual Deletion | 4.5% (18) |
| | Never Uninstalled | 20.75% (83) |
| | Other | 0% (0) |
| | Unsure | 3% (12) |
| **Understanding of KSA Functions** | Yes | 40.75% (163) |
| | No | 59.25% (237) |
| **Knowledge of KSA Activation** | Always Active | 14.75% (59) |
| | Active on Access | 60.75% (243) |
| | Active on Specific Access | 20.75% (83) |
| | Unsure | 3.75% (15) |
| **Perceived Helpfulness** | Not Helpful | 11.25% (45) |
| | Slightly Helpful | 17.75% (71) |
| | Moderately Helpful | 27% (108) |
| | Somewhat Helpful | 38.25% (153) |
| | Very Helpful | 5.75% (23) |

asymmetric keys, ostensibly safeguarding its integrity from unauthorized access. However, attackers can maliciously manipulate the data by hooking or binary patching it while it's still in plaintext before encryption within KSA's internal processes. To validate this vulnerability, we conducted experiments by intercepting the plaintext data pre-encryption and verifying the modifications post-decryption on a demo site. Our findings underscore the system's inability to effectively block transactions originating from countries flagged as high-risk by individual banks or to prevent attackers engaging in abnormal transactions from concealing their identities.

## D.2 Accredited Certificate Key Leakage

We also found that a certain KSA (PRODUCT B) has a vulnerability in its implementation that lead to the leakage of private keys of Accredited Certificate. These certificates are used for authentication, and thus, they can be used to identify the user. Certificate management KSA includes an API for copying certificates. With this API, users can copy local certificates to the browser's storage. During this process, users must input the certificate password for validation before transmitting the certificate key pair to the browser. However, we discovered that PRODUCT B transmits the certificate key pair to the browser even if password validation fails. Although this private key is encrypted with an additional password, if we combine this vulnerability with the aforementioned keylogging attack, we can steal the user's private key and impersonate the user. This allows us to bypass the certificate-based authentication and gain full access to the user's bank account, leading to serious financial damage. Notably, such data are originally inaccessible via the browser due to the sandbox but become accessible through KSA.

## D.3 Other Issues

**Flaws in origin check.** We also found several issues in origin checks in KSA. Firstly, some KSA verify the Origin just once after booting up the computer, subsequently permitting unrestricted access. This KSA operates with normal privileges until they successfully verify the Origin, at which point they load high-privileged software. In this case, attackers can bypass Origin checks by redirecting to allowed web pages and then proceed with the attack.

Secondly, some security vendors uses cryptography incorrectly for origin checks. In particular, KSA validates origin by comparing it with a list of whitelisted origins. This list is encrypted with a secret key known only to the security vendor. Unfortunately, KSA uses plain RSA encryption using PKCS #1 padding, which does not provide any integrity. As a result, attackers can perform brute-force attacks (within a reasonable time) to break this origin check.

**Install management KSA.** In this paper, we focused on mandatory KSA, but in South Korea, there are many other KSA, which can also lead to various security issues. One of them is the install management KSA. To use services, users must install multiple KSA, which can be challenging for those unfamiliar with computers. Thus, there is another KSA called an integrated install management KSA. This receives a list of KSA from customer companies and installs them for users. We also have analyzed this KSA and found that it does not validate the code signatures of downloaded programs. As a result, if the download server is compromised and malicious code is installed, this KSA does not provide defense against it. This vulnerability also demonstrates a lack of consideration for security in the design of this KSA.