Using Inputs and Context to Verify User Intentions in Internet Services

He Shuang, Wei Huang, Pushkar Bettadpur, Lianying Zhao, Ivan Pustogarov, David Lie
University of Toronto
{he.shuang,wei.huang,pushkar.bettadpur}@mail.utoronto.ca
{lianying.zhao,i.pustogarov,david.lie}@utoronto.ca

ABSTRACT

An open security problem is how a server can tell whether a request submitted by a client is legitimately intended by the user or fakes by malware that has infected the user's system. This paper proposes Attested Intentions (AINT), to ensure user intention is properly translated to service requests. AINT uses a trusted hypervisor to record user inputs and context, and uses an Intel SGX enclave to continuously verify that the context, where user interaction occurs, has not been tampered with. After verification, AINT also uses SGX enclave for execution protection to generate the service request using the inputs collected by the hypervisor. To address privacy concerns over the recording of user inputs and context, AINT performs all verification on the client device, so that recorded data is never transmitted to a remote party.

ACM Reference Format:
He Shuang, Wei Huang, Pushkar Bettadpur, Lianying Zhao, Ivan Pustogarov, David Lie. 2019. Using Inputs and Context to Verify User Intentions in Internet Services. In *10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '19), August 19–20, 2019, Hangzhou, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3343737.3343739

1 INTRODUCTION

A long-standing problem in Internet systems security is how a service can ensure whether a request received from a remote client[1] is user-intended. We begin by reviewing some of these solutions, and show how they fall prey to straightforward attacks commonly used by Internet miscreants today.

Consider an online payment service where Alice wishes to pay $100 to Bob. An attacker, Mallory, wishes to forge a request that will cause Alice to pay $1000 to Mallory instead. The most basic attack is for Mallory to steal login/password credentials from Alice and send a request for such a payment to the service. This attack is prevented by having appropriate authentication mechanisms. Traditional username/password authentication can be further enhanced by multi-factor authentication, which adds additional lines of defence beyond the username and password.

However, things change if Mallory is able to find a vulnerability in Alice's client that allows Mallory to install her own malicious software (i.e., OS-level malware). This gives Mallory the capability to perform client-side attacks. A strong attacker may attempt to tamper with the execution of software to affect the integrity of the service request; we call these attacks **Execution Tampering**. Execution tampering can occur at any level of the system (kernel or userspace), but the salient points are 1) the attack is stealthy and 2) it works even if the user acts correctly (i.e., all her actions are consistent with her intentions). Execution tampering requires the malware to gain privileges on the victim system, a situation not always possible, and can lead to scenarios wherein an attacker can only alter the display but not the code execution. Our second type of attack is **Context Forgery**, where the user interface (UI) is altered in a way to affect user perception and lure them into carrying out unintended actions. To give an example, as Alice is entering $100 as the transaction amount, the attacker intentionally blocks one zero and displays $10, Alice might be fooled in believing that she has only entered $10 instead of $100. If Alice enters another zero, the malicious attacker can carry out a transaction of amount $1000 instead of user intended $100. In this case, Alice's input occurs in an *altered context* under her *impaired perception*. The salient points for context forgery are that 1) user interaction does not properly follow user intention and 2) it works even with protected execution.

There exist a number of proposals to defeat such client-side attacks. While many operate purely at the OS-level

[1]In this paper, we use the term *client* to denote any computing device, including smartphones, PCs and even embedded/IoT devices.

APSys '19, August 19–20, 2019, Hangzhou, China

ACM ISBN 978-1-4503-6893-3/19/08…$15.00
https://doi.org/10.1145/3343737.3343739

(i.e., click-jacking defenses) [16, 31], and thus would fail against an attacker with OS-level privileges, a number of defenses assume a compromised OS. For example, trusted computing [2, 4, 17, 23–25, 34] allows clients to attest the integrity of software components to a remote party. However, they are incomplete as they cannot assert anything about the integrity of inputs given to the attested components, or the context under which the user gave those inputs.

Still other proposals attempt to combine trusted computing with checking for external inputs [10, 15]. However, these proposals do not bridge the *semantic gap* [8, 18] that exists between raw inputs collected by the hardware and the *context* under which the user generated the inputs. For example, NAB checks that there were external inputs around the time a user request is generated, opening the door for a context forgery attack where the attacker harvests the user activity on an unrelated application, while the malware sends the forged request on a security-critical application (i.e., payment app). This has further led to proposals that attempt to better link user input with an outgoing request. For example, Gyrus [19] ensures that an outgoing request contains input typed by the user. However, this is insufficient as the context where the input occurs is *not* validated (e.g., only user-typed data is checked). For example, a context forgery attack can be done by simply swapping "OK" and "Cancel" buttons. Fidelius [12] and VButton [20] implement a trusted display in parallel with an untrusted display; the context where a user sits consists of both displays. However, studies have shown that such approaches are ineffective, and an attacker that is only able to control the untrusted display can still affect the user's perception [13].

From this, a correct solution requires that a) the execution of code should be protected from tampering, b) the inputs provided by the user should be bound to the request and c) the context under which the user provides the inputs must be verified. No previous work has done all three and none are able to do (c) effectively.

We propose Attested Intentions (AINT, we use AINT to mean the system and Attested Intention to mean a verified request), which attests to not only the integrity of request generation, but also the coherence of the *input* and its *context*. For example, to prevent forgery of a payment request, the input could be the keystrokes of the user filling out the payment form and the mouse moves to click the "Pay" button. Similarly, the user context could be screenshots showing the interaction of the user with the website. This, when combined with information about the execution integrity of the client, allows the service to determine whether requests are legitimate or forged. AINT protects against execution tampering and context forgery at the same time. To protect against execution tampering, a trusted hypervisor collects user inputs and processes them inside a userspace Intel SGX enclave, saving the necessity to trust the entire software stack. To protect against context forgery, AINT validates what a user sees against what is expected by the service and binds the result to the request. With AINT, an attacker can no longer forge a request, send the request while the user is interacting with an unrelated application, or alter the context where the request is generated.

AINT verifies the interaction locally on the user's client device and then cryptographically attest to the server that the Attested Intention generated on the client device is consistent with the request sent by the client device. This allows the service to receive assurances that the request is legitimate without having access to the data collected by AINT, which remains local to the client device.

Contributions. We make the following contributions.

- We identify two types of attacks: execution tampering and context forgery, which can be used to violate user intention and send altered service requests.
- We propose Attested Intentions as a solution to both execution tampering and context forgery that ensures any service request received by a service provider, is user-intended.
- We take user privacy into account by not sharing any private information with any remote party.

2 THREAT MODEL AND GOALS

We assume a powerful remote attacker who has full control of the OS. We further assume our attacker has the ability to read or write any unencrypted memory and to execute arbitrary code on the client. However, we assume the service deploys multi-factor authentication, meaning that OS compromise does not imply that an attacker can automatically impersonate the user. We do not consider denial of service attack because a compromised OS can always stop the user from making requests.

Since AINT requires Intel SGX and assumes I/O data from the hardware carries user intention, we assume that computer hardware including the processor and peripherals are not compromised. AINT also requires the use of a hypervisor, therefore a hypervisor is trusted.

Even though AINT aims to ensure proper user perception, we still require user *cautiousness* so that the user will carry out only intended actions under properly generated context. A careless user can claim any request to be unintended. Further, we trust the service providers that they do not send malicious data to the users.

Phishing, where user perception may be affected by an impersonating service, cannot be directly addressed, since AINT is application initiated. An impersonating application can simply skip the initiation of the AINT session. However, we note that even though AINT does not provide anti-phishing

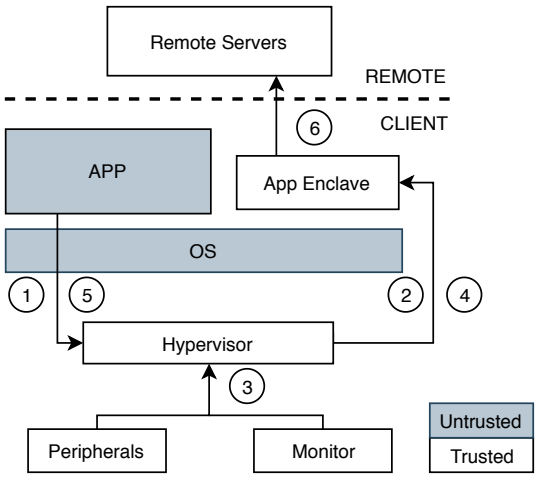

Figure 1: AINT overview.

protection to the user, it still provides the anti-bot guarantees to the service.

Goals. For a request received by a service, AINT ensures the request is 1) *user-intended*: a physical human initiated and requested the service. 2) *integrity-protected*: untampered during generation and transmission. 3) *authenticity-protected*: originates from the same machine where AINT resides.

3 DESIGN

AINT acquires three pieces of information about the request: a) inputs that the user provided to the client, b) the context under which those inputs were provided and c) an execution guarantee that user inputs were correctly transformed into a service request. AINT needs all three to achieve its goals. (a) and (b) together conclude that the user, under a proper context, intended a particular action. By combining with (c), we can confirm that a user-intended action is properly translated into a request. We note that (a) plus (c) defeats Execution Tampering, while (b) plus (c) defeats Context Forgery. To sum up, AINT's design involves three requirements:

- **Inputs**: capture authentic user inputs. Section 3.2.
- **Context**: capture and validate context user sees against service-provided data and user inputs, and bind context to the service request. Section 3.3.
- **Request Generation**: a request is generated with authentic inputs and the execution is integrity protected. Section 3.4.

3.1 System Overview

The overview of AINT is shown in Figure 1. Before AINT runs, the service provider deploys an App Enclave on the client. App Enclave, after initiation, immediately attests its integrity to the remote server. The service data is sent to the application from the service provider. We define *service data* as files that service providers send to users for immediate interaction. In minimum, that includes 1) the construction of user context and 2) the handling of user interaction. In the case of a browser application, service data includes HTML, JavaScript and CSS files.

The hypervisor waits for the application to trigger an AINT secure session ①. To do so, the application must supply service data, as received from the service provider. The service data is sent to App Enclave ②, which checks its authenticity and integrity by verifying its signature. During the secure session, the hypervisor records user inputs from peripherals and user context from display monitors ③, signs and sends them to App Enclave ④. The App Enclave then continuously verifies that the user context is consistent with the expected user context by verifying two things: 1) the presentation of service data through the use of an AINT function (Section 3.3.2) and 2) the presentation of input previously collected by the hypervisor (Section 3.3.3).

The application must also notify AINT in order to finish the secure session ⑤. At the end of the session, if it can determine that no context tampering has occurred, AINT will carry out request generation inside App Enclave and send the final attested service request to the remote server ⑥. Since the remote server knows that the request comes from a previously attested enclave, it gains confidence that the service request follows the user intention.

3.2 User Inputs

We consider all data received locally via user interaction as user inputs. AINT makes use of the input in two ways, 1) context verification and 2) generation of the final service request. AINT not only ensures the integrity and authenticity of input data when they are being collected, it must also ensure input data remains integrity protected after collection.

AINT requires a way for triggering a secure session, such as a hypercall from a client application. This hypercall also provides the signed raw service data necessary for AINT to check the user's context and as a part of the service request specification, indicates what inputs need to be collected. In our payment example, those inputs would include keyboard inputs (i.e., when the user enters the name of the payee and the amount), as well as mouse movements and clicks (i.e., if the user is instead required to select from a list of payees, and when the user clicks the "Pay" button to submit the form). These inputs are recorded and form the content of the Attested Intention for this payment request.

Since user inputs are completely captured by the hypervisor, to allow App Enclave to verify the trustworthiness of the hypervisor and to perform trusted hypervisor boot and attest to the integrity of the hypervisor, AINT uses a trusted computing mechanism, such as Intel TXT or AMD SVM with

a TPM [36, 37].

3.3 Context

One of the novel aspects of AINT is the collection and validation of context to help verify a request. We define context as the rendering of service data, as well as the rendering of the user inputs (i.e., keyboard input being echoed back to the user). Altered user inputs may affect user perception as shown previously (e.g., failing to echo a zero in our example). Collecting and validating inputs allows AINT to stop context forgery attacks where an attacker splits the context in which a user generates a request from the actual request sent by the client and guarantees *what the user sees is what is supposed to be seen*. Context verification is done securely inside App Enclave and AINT ensures that requests are only generated from a proper context receives attestation, which effectively binds context to the request.

3.3.1 ***Context Acquisition.*** Naïvely, one might consider a screen capture of what is on the screen when the user clicks the "Pay" button, to be consistent with what the user sees and correctly capturing the intention. However, this is overly simplistic and does not take into account the *temporal integrity* of the display. In particular, a screenshot only captures the screen at a particular moment in time. Given the speed of computer systems, a compromised browser or web page could easily show one screen to a user (i.e., one where the Alice is paying $1 to Bob) and then just as the user is about to click the "Pay" button, change it to a screen consistent with the forged request (i.e., pay $100 to Mallory) so that it appears to App Enclave that Alice intended to pay $100 to Mallory instead of $1 to Bob. Further, the browser could switch the screen back just after the click faster than a human Alice could detect. As a result, in this scenario, it is necessary to send a continuous video capture of the user filling in the appropriate form in the payment application to correctly certify the request.

Like user inputs, user context is specific to the type of interaction being verified and can include other passive measurements about what the user perceives at the time. For example, on a smartphone client, this could even include the user's location, other running applications, etc. to verify the user's context. We note that while a user's screen contents and location contain private information, the App Enclave verifies this entirely on the device, thus ensuring that privacy is protected. We also note that while SGX is specific to Intel processors, similar functionality is provided by TrustZone on ARM processors, which are more commonly found on smartphones.

There are various methods for acquiring user context, ranging from pure software approach such as hypervisor-based virtualized GPU to dedicated hardware such as an HDMI grabber, and co-designs such as Nvidia ShadowPlay [28] and AMD ReLive [3]. These approaches vary in terms of cost, slow down to the machine and TCB size.

3.3.2 ***Service Data Verification.*** The verification of service data is done inside App Enclave, through the use of an AINT function, a function that determines if a rendered display is rendered properly. The AINT function is application-specific and provided by the service. The trusted hypervisor must submit context and inputs, while the application must submit raw service data to App Enclave. Before checking, App Enclave must verify the integrity and authenticity of these inputs, for example by checking that the HTML is signed by the server, and context and that user inputs are signed by the hypervisor.

To verify displayed context, our current approach is to conduct a pixel-by-pixel comparison between service pre-rendered context with the captured context. However, many applications, including browsers, may not be entirely deterministic, and even a simple HTML page may have external dependencies (i.e., other pages or components it must load) that can affect its appearance. In addition, local optimization such as Windows ClearType [26], a form of anti-aliasing, adds additional non-deterministicity and renders the enclave's pixel-by-pixel comparison ineffective. The verification model that App Enclave uses must be able to distinguish between innocuous and malicious differences in the recorded context and client-generated request, and the context and request expected by the model. In the case of a payment application, the user interface is likely deterministic enough that a simple image similarity comparison [5, 32] of frames may be sufficient to check that the user context was rendered correctly. Predictable differences, such as current time or content from external dependencies, can be specifically ignored by the execution model.

Another option for the AINT function is to run machine learning-based image segmentation to extract the objects from the context, and compare their size and position with the specification from the HTML files from service providers. However, previous works have shown that machine learning techniques are vulnerable to adversarial attacks, especially image-based models [7, 30].

3.3.3 ***Input Verification.*** With proper context verification, input verification can be carried out similarly. Instead of periodically validating the service data rendering, input verification is triggered whenever a user input event occurs and maintained afterwards.

Implementing AINT using SGX and a trusted hypervisor faces the same semantic gap challenges that previous systems that used hypervisors for security faced [8, 18]. However, while previous systems have primarily dealt with inferring OS-level information without having to trust the OS [22],

the challenge here is to infer what context the user sees and what input they are entering. As an example, consider the case of mouse input. Mouse signals are not transmitted as absolute coordinates, but as relative offsets to the current mouse position, which is maintained by the OS. Moreover, since the OS is not trusted, the hypervisor cannot trust the position of the mouse reported by the OS. While the hypervisor could continuously monitor relative updates from the mouse hardware and maintain its own mouse position, there are many events that could cause the hypervisor's view of the mouse to become desynchronized from that of the OS's, which is what is perceived by the user. For example, certain keyboard shortcuts may cause the mouse cursor to snap to a new position. Without full access to the state of the OS graphic user interface, the hypervisor would not be able to compute the new mouse position. We propose a general approach where the hypervisor requests the OS to force the internal state to a well-known value and thus remove the semantic gap for the duration of an AINT session, an approach we call *semantic synchronization*. For example, for the mouse, the hypervisor can ask the OS to reset the mouse pointer to a known location at the beginning of the secure session. A malicious OS that ignores this request will cause a mismatch between how the inputs are interpreted by the hypervisor and the actions recorded in the user context, which will be detected by the verification algorithm.

Not all input methods can be solved with semantic synchronization; certain input types require access to the captured context. For instance, clipboard paste is an input method that inserts previously copied text (we assume text for simplicity). To accurately capture the semantic meaning (text being pasted) of the paste action, at the time of copy, AINT must capture the currently highlighted text. Assuming copy always occurs while AINT is running (within or outside of the secure session), AINT must examine every frame of the captured context prior to the copy action searching for highlighted text. This is not only complex and error-prone, but also introduces a large amount of code. Even with that, certain events, such as web page's "Click to Copy" JavaScript, may cause AINT's view of the copied text to be desynchronized from the OS's. If AINT were to properly handle all types of input methods such as copy & paste, backspace, auto-complete and virtual keyboard, it will hugely increase the complexity and the trusted computing base of AINT. Therefore, AINT currently does not implement checks for these input methods and prohibits the use of them during a secure session.

3.4 Request Generation

The request generation code is part of the service data submitted by the application, but validated by AINT before execution. For instance, in the payment service example, the request generation code collects fields such as transaction amount and recipient and generates a service request with signed values for server processing (similar to the JavaScript handlers). AINT takes inputs from the hypervisor and performs execution inside an Intel SGX enclave. This defeats execution tampering without trusting the entire software stack.

One might think that instead of generating the service request inside App Enclave, it can outsource request generation to the application and only verify generated request [21] against captured inputs and context. The intuition driving this is that verifying an execution consists of less code than doing the execution, thus reducing the trusted computing base (TCB) of the App Enclave. However, the verification proposed in [21] is restricted to only simple range checks, and it can be challenging to verify all inputs accurately and comprehensively, thus limiting the range of functions that AINT can check. For instance, for some complex input processing functions, the only sure check may be to redo the entire computation, thus negating any possible TCB savings. In contrast, running request generation in the enclave achieves security, simplicity and wider applicability. Therefore, with the trade-off between applicability and TCB, AINT chooses applicability.

3.5 Privacy

By processing user inputs and context locally, App Enclave ensures that this information does not leave the client device. At the same time, AINT must be entirely trusted by the server since the server never sees the raw user input and context. This means that App Enclave must be mutually trusted by both the user and the server. This requirement is satisfied by trusted computing, and in our example, by SGX attestation, but the requirement could be satisfied by a number of trusted computing primitives such as TXT or TrustZone. We envision that the server specifies App Enclave, as the server must develop the content-specific AINT function. To allow users to trust server's code, a trusted third party should verify all code inside App Enclaves to ensure service developed code are indeed privacy preserving.

Previous works have proposed using trusted computing as a trusted third party to verify values [1]. Most recently, Glimmer proposed to use SGX to verify and cryptographically blind information so that it is not leaked to Internet services [21]. However, with a trusted hypervisor, AINT is able to verify human interactions and thus provide much stronger guarantees than Glimmer, which without trusted user inputs and context, can only sanity check the validity of user inputs (for example by checking that the inputs fall within a valid range).

4 TRUSTED COMPUTING BASE (TCB)

To maintain a small TCB for the hypervisor, while there exist minimal hypervisors [33] and even formally verified hypervisors [37], whose implementations may contain 500K lines of code or less, such hypervisors typically do not virtualize hardware to keep the TCB small, but instead pass it through to drivers in the guest OS. However, in dealing with stateful devices, it is tempting to simply duplicate the entire driver stack inside the hypervisor so that it can replicate the state changes in the OS that result from inputs from the hardware device. Some device drivers may contain millions of lines of code, bloating the TCB of an otherwise small hypervisor [38]. Instead, we propose that the hypervisor intercepts and emulates only the relevant portions of the drivers. For example, in complex I/O such as USB, the hypervisor does not need to include the entire USB stack, but just the portions that interface with the USB hardware to access the raw communication data. In combination with semantic sync events, we believe that the semantic gap can be bridged without large increases in the TCB of the trusted hypervisor. To do so, AINT statically splits the driver [14, 38], and outsources non-critical operations such as bus enumeration and device hot-plug to the general-purpose OS and verifies the result.

AINT TCB includes drivers, as AINT depends on them to intercept and record user input. Moving driver functionality required for AINT into the hypervisor reduces the TCB, as vulnerabilities in the driver code that is not critical to AINT can no longer impact AINT security guarantees [27]. For example, vulnerabilities in unrelated driver functionality, such as power management, or in other unrelated drivers [9, 35] cannot affect the portion of code that has been moved into the hypervisor to implement AINT.

5 USE CASES

In addition to ensuring the secure flow of user intention to a service request, we detail some use cases of AINT.

CAPTCHA is currently widely used to distinguish bots from humans, but CAPTCHA response is not bound to the context, allowing attacks such as CAPTCHA farms [11]. To make matters worse, CAPTCHAs make systems less usable: according to Bursztein et al. [6], in some instances, humans may only be able to correctly solve visual CAPTCHAs as low as 61% of the time, and audio CAPTCHAs 31% of the time. AINT can be used to implement a user-requirement-free CAPTCHA replacement. Specifically, AINT tightly binds user context and physical activities to a particular machine, and thus the network request from the machine. This defeats CAPTCHA farms, where request are split from the CAPTCHA response. Bots cannot harvest user activities, because AINT enforces the right context when a request is generated. Since all user activities are examined locally on the user device, there is no privacy concern compared to transitional CAPTCHA services [29].

Remote Invigilation is a type of activity where a test taker is being invigilated remotely. The invigilator must ensure the absence of unauthorized aids, but a test taker may attempt to cheat by modifying the local software, such as redirecting the live video feed to a previous recording. Even though AINT requires users to be cautious and acts accordingly to the context, we note that it does not conflict with the assumption here because, so long as the other assumptions hold (i.e., trustworthiness of the hardware)—AINT's goal, in this case, is to correctly verify user interactions (is the test taker trying to cheat ?). An invigilator has to require 1) a secure system state, meaning the absence of malware 2) authentication of the test taker's action and 3) the absence of unauthorized aids. AINT naturally satisfies the first two requirements due to its threat model and the ability to attest authentic user inputs. The third requirement is satisfied by implementing plagiarism detection inside the AINT function. For instance, the AINT function checks the context for any unauthorized window or window switching. AINT has several advantages: 1) reliable software level protection, AINT can protect itself from OS level threat and 2) the computation happens on client devices saving server resources. However, AINT requires the trustworthiness of the hardware and thus is unable to provide any guarantee if the candidate compromises the hardware.

6 FUTURE WORK AND CONCLUSION

Our current efforts are focused on generalizing the design and implementation of the AINT function to cover a wide range of scenarios, and addressing the semantic gap to reliably record user inputs and context without having to duplicate the complete driver stack in the trusted hypervisor.

In conclusion, we detailed two types of attacks, execution tampering and context forgery, that can violate user intention and alter service requests. We proposed Attested Intention as a solution to both attacks, with the idea of using user inputs and context to verify service requests. This enables a remote service provider to gain assurance that the request is user intended. AINT runs entirely on user client to preserve privacy. AINT leverages a small trusted hypervisor and Intel SGX to provide assurance even under full client OS compromises.

ACKNOWLEDGEMENT

Special thanks to Piaoyao Shi for her thoughtful comments on an earlier draft of the paper. The authors would also like to thank Petros Maniatis for early inspiration for this work. This work is supported in part by an NSERC Discovery Grant (RGPIN-2018-059) and by an OGS Graduate Scholarship.

REFERENCES

[1] Martín Abadi. 2004. Trusted Computing, Trusted Third Parties, and Verified Communications. In *Security and Protection in Information Processing Systems*, Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang (Eds.). Springer US, Boston, MA, 291–308. https://users.soe.ucsc.edu/~abadi/Papers/verif.pdf

[2] AMD. 2018. AMD64 Architecture Programmer's Manual Volume 2: System Programming. Retrieved Jan 17, 2019 from https://www.amd.com/system/files/TechDocs/24593.pdf

[3] AMD. 2018. How to Capture and Stream Gameplay Using Radeon ReLive. Retrieved July 6, 2019 from https://www.amd.com/en/support/kb/faq/dh-023

[4] ARM. 2009. Arm Security Technology - Building a Secure System using TrustZone Technology. Retrieved Jan 16, 2019 from http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/ch01s02s01.html

[5] Ashwin Swaminathan, Yinian Mao, and Min Wu. 2006. Robust and secure image hashing. *IEEE Transactions on Information Forensics and Security* 1, 2 (June 2006), 215–230. https://doi.org/10.1109/TIFS.2006.873601

[6] Elie Bursztein, Steven Bethard, John C. Mitchell, Dan Jurafsky, and Céline Fabry. 2010. How Good are Humans at Solving CAPTCHAs? A Large Scale Evaluation. In *IEEE Symposium on Security and Privacy*. Oakland, CA, USA. http://ieeexplore.ieee.org/document/5504799

[7] Nicholas Carlini and David A. Wagner. 2016. Towards Evaluating the Robustness of Neural Networks. *CoRR* abs/1608.04644 (2016). arXiv:1608.04644 http://arxiv.org/abs/1608.04644

[8] Peter M. Chen and Brian D. Noble. 2001. When virtual is better than real [operating system relocation to virtual machines]. In *Proceedings Eighth Workshop on Hot Topics in Operating Systems*. 133–138. https://doi.org/10.1109/HOTOS.2001.990073

[9] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. 2001. An Empirical Study of Operating Systems Errors. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*. ACM, New York, NY, USA, 73–88. https://doi.org/10.1145/502034.502042

[10] Weidong Cui, Randy H. Katz, and Wai-tian Tan. 2005. BINDER: An Extrusion-based Break-In Detector for Personal Computers. In *Proceedings of the 2005 USENIX Annual Technical Conference*. USENIX Association, Berkeley, CA, USA. https://www.microsoft.com/en-us/research/publication/binder-an-extrusion-based-break-in-detector-for-personal-computers/

[11] Dancho Danchev. 2018. Inside India's CAPTCHA solving economy. Retrieved July 6, 2019 from https://www.zdnet.com/article/inside-indias-captcha-solving-economy/

[12] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Brandon, Dillon Franke Franke, Forest Fraser, Gaspar Garcia, Eric Gong, Hung T. Nguyen, Taresh K. Sethi, Vishal Subbiah, Michael Backes, Giancarlo Pellegrino, and Dan Boneh. 2019. Fidelius: Protecting User Secrets from Compromised Browsers. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA. https://doi.org/10.1109/SP.2019.00036

[13] Yanick Fratantonio, Chenxiong Qian, Simon P Chung, and Wenke Lee. 2017. Cloak and dagger: from two permissions to complete control of the UI feedback loop. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1041–1057. https://ieeexplore.ieee.org/document/7958624

[14] Vinod Ganapathy, Matthew J. Renzelmann, Arini Balakrishnan, Michael M. Swift, and Somesh Jha. 2008. The Design and Implementation of Microdrivers. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*. ACM, New York, NY, USA, 168–178. https://doi.org/10.1145/1346281.1346303

[15] Ramakrishna Gummadi, Hari Balakrishnan, Petros Maniatis, and Sylvia Ratnasamy. 2009. Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. USENIX Association, Berkeley, CA, USA, 307–320. http://dl.acm.org/citation.cfm?id=1558977.1558998

[16] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schecter, and Collin Jackson. 2012. Clickjacking: Attacks and Defenses. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX, Bellevue, WA, 413–428. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/huang

[17] Intel. 2010. Intel® Trusted Execution Technology: White Paper. Retrieved Nov 15, 2018 from https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html

[18] Bhushan Jain, Mirza Basim Baig, Dongli Zhang, Donald E. Porter, and Radu Sion. 2014. Sok: Introspections on Trust and the Semantic Gap. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 605–620. https://ieeexplore.ieee.org/document/6956590

[19] Yeongjin Jang, Simon P Chung, Bryan D Payne, and Wenke Lee. 2014. Gyrus: A Framework for User-Intent Monitoring of Text-based Networked Applications.. In *Proceedings of the 2014 Network and Distributed System Security Symposium*. https://www.ndss-symposium.org/ndss2014/programme/gyrus-framework-user-intent-monitoring-text-based-networked-applications/

[20] Wenhao Li, Shiyu Luo, Zhichuang Sun, Yubin Xia, Long Lu, Haibo Chen, Binyu Zang, and Haibing Guan. 2018. VButton: Practical Attestation of User-driven Operations in Mobile Apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*. ACM, New York, NY, USA, 28–40. https://doi.org/10.1145/3210240.3210330

[21] David Lie and Petros Maniatis. 2017. Glimmers: Resolving the Privacy/Trust Quagmire. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS '17)*. ACM, New York, NY, USA, 94–99. https://doi.org/10.1145/3102980.3102996

[22] Lionel Litty, H Andrés Lagar-Cavilla, and David Lie. 2008. Hypervisor Support for Identifying Covertly Executing Binaries. In *Proceedings of the 17th Conference on Security Symposium (SS'08)*. USENIX Association, San Jose, CA, 243–258. https://dl.acm.org/citation.cfm?id=1496728

[23] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB Reduction and Attestation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP '10)*. IEEE Computer Society, Washington, DC, USA, 143–158. https://doi.org/10.1109/SP.2010.17

[24] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. 2008. Flicker: An Execution Infrastructure for Tcb Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (Eurosys '08)*. ACM, New York, NY, USA, 315–328. https://doi.org/10.1145/1352592.1352625

[25] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13)*. ACM, New York, NY, USA, Article 10, 1 pages. https://doi.org/10.1145/2487726.2488368

[26] Microsoft. 2017. Microsoft ClearType overview. Retrieved July 6, 2019 from https://docs.microsoft.com/en-us/typography/cleartype/

[27] Subhas C. Misra and Virendra C. Bhavsar. 2003. Relationships Between Selected Software Measures and Latent Bug-density: Guidelines for

Improving Quality. In *Proceedings of the 2003 International Conference on Computational Science and Its Applications: PartI (ICCSA'03)*. Springer-Verlag, Berlin, Heidelberg, 724–732. http://dl.acm.org/citation.cfm?id=1756748.1756832

[28] Nvidia. 2017. Record and Capture your Greatest Gaming Moments. Retrieved July 6, 2019 from https://www.nvidia.com/en-us/geforce/geforce-experience/shadowplay/

[29] Lara O'Reilly. 2015. Google's new CAPTCHA security login raises 'legitimate privacy concerns'. Retrieved Dec 7, 2018 from https://www.businessinsider.com.au/google-no-captcha-adtruth-privacy-research-2015-2

[30] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. ACM, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[31] Andrea Possemato, Andrea Lanzi, Simon Pak Ho Chung, Wenke Lee, and Yanick Fratantonio. 2018. ClickShield: Are You Hiding Something? Towards Eradicating Clickjacking on Android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1120–1136.

[32] Ramarathnam Venkatesan, S.-M Koon, Mariusz Jakubowski, and Pierre Moulin. 2000. Robust image hashing. In *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, Vol. 3. 664–666 vol.3. https://doi.org/10.1109/ICIP.2000.899541

[33] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. 2009. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '09)*. ACM, New York, NY, USA, 121–130. https://doi.org/10.1145/1508293.1508311

[34] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Haining Wang. 2015. TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 367–378. https://doi.org/10.1109/DSN.2015.11

[35] Michael M. Swift, Brian N. Bershad, and Henry M. Levy. 2003. Improving the Reliability of Commodity Operating Systems. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 207–222. https://doi.org/10.1145/945445.945466

[36] Trusted Computing Group. 2016. TPM 2.0 Library Specification. Retrieved Nov 15, 2018 from https://trustedcomputinggroup.org/resource/tpm-library-specification/

[37] Amit Vasudevan, Sagar Chaki, Limin Jia, Jonathan McCune, James Newsome, and Anupam Datta. 2013. Design, Implementation and Verification of an eXtensible and Modular Hypervisor Framework. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 430–444. https://doi.org/10.1109/SP.2013.36

[38] Zongwei Zhou, Miao Yu, and Virgil D. Gligor. 2014. Dancing with Giants: Wimpy Kernels for On-Demand Isolated I/O. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA, 308–323. https://doi.org/10.1109/SP.2014.27