Is a Bot at the Controls? Detecting Input Data Attacks

Travis Schluessler
travis.schluessler@intel.com

Stephen Goglin
stephen.d.goglin@intel.com

Erik Johnson
erik.j.johnson@intel.com

Intel Corporation
2111 NE 25th Ave
Hillsboro, OR 97123

ABSTRACT

The use of programmatically generated input data in place of human-generated input data poses problems for many computer applications in use today. Mouse clicks and keyboard strokes can automatically be generated to cheat in online games, or to perpetrate click fraud. The ability to discern whether input data was computationally generated instead of created by a human input device is therefore of paramount importance to these types of applications. This paper describes a method for detecting input data that was computationally modified or fabricated. This includes detecting data that was not directly generated by a physical human input device such as a keyboard or mouse. A prototype of this system was built on existing hardware and was shown to be effective at detecting attacks on a real application. This detection method is capable of addressing the majority of input-based attacks currently in use. When used in conjunction with a trusted peripheral, it offers a robust mechanism for ensuring a computer is not at the controls.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and Application-based Systems

General Terms

Security

Keywords

Security, Online Games, Cheating, Cheat Detection

1. INTRODUCTION

A growing number of computing applications are faced with a fundamental question critical to the privacy, authenticity, and integrity of their operations: "Is a human in control, or is it a computer?" Answering this question is important because attacks can be waged against computing systems by programmatically modifying input data that was supposed to come from a human using an input device. For example, users can cheat in online games by having software generate input to aim their weapons or play the game for them. Click fraud can be perpetrated by using software to generate mouse clicks on online advertisements that deplete a competitor's advertising budget or win an online auction. As these attacks spread to more types of applications, the financial impact becomes substantive [13]. Correspondingly, a method of detecting such attacks is of great importance.

Several methods have been employed to mitigate input data attacks including CAPTCHAs [1], and anti-cheat software [21]. These methods face limitations. For example CAPTCHAs have an inherent performance limitation [6] that constrains their use to non-real-time applications. A complication all methods face when attempting to address input data attacks is that the attacker typically has physical control of the system being attacked.

This research demonstrates a method capable of ensuring input data enters a system through a physically present human input device (HID). The method can detect when data is illicitly modified prior to its use. By reliably comparing input data consumed by an application to data generated by input hardware, attacks that generate or modify input data can be detected in a manner that is independent of the operating system and software stack. This forces attackers to move to hardware based attacks.

The input monitoring system outlined in this paper operates on the premise that the input data generated by HIDs must be the same as the input data consumed by the software application. If the two data streams differ, some form of illicit modification, insertion, or deletion occurred. This premise makes attack detection straightforward: in order to detect an attack, the input monitoring system simply needs to compare the two data streams and if a discrepancy is detected, notify a remote service provider. This type of detection approach is sometimes referred to as catching the system in a lie [14]. The comparison and notification operations must be performed in a reliable, tamper-resistant manner, meaning the platform owner cannot interfere with the operations. These concepts are shown in Figure 1. This input monitoring system can be built with minor extensions to hardware and firmware that already exists in many PCs in use today.

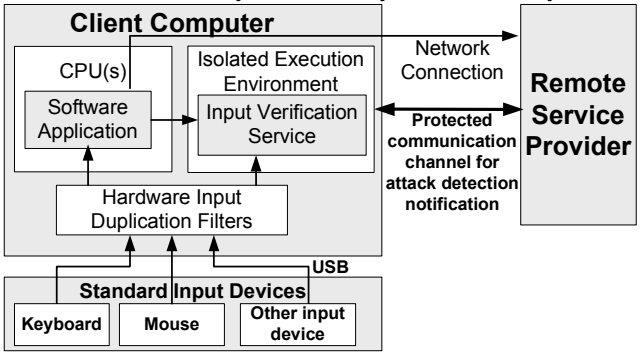

Figure 1: System Architecture

Netgames'07, September 19-20, 2007, Melbourne, Australia

The nature of this problem is described in section 2 along with an analysis of existing solutions. Section 3 explains how the system operates. Section 4 presents an evaluation of a prototype built on existing PC hardware and also describes the system's limitations. The future direction of the work is outlined in Section 5 and conclusions are presented in Section 6.

2. PROBLEM STATEMENT

Input data modification attacks can be used to achieve illicit effects on software programs such as cheating in online games and auctions, and click fraud. A primary challenge in protecting against these attacks is that the perpetrator is often the system owner, with physical access to the system. Despite this, a number of defense mechanisms have evolved that try to address aspects of the problem. This section gives examples of these attacks and the existing methods for protecting against them.

2.1 Input Attack Examples

2.1.1 Online Game Cheating

Winning in online games is no longer just about fun, it can now translate into real currency. Money can be made in online gaming through gambling, real money trades of virtual commodities [26], and leagues awarding prize money. This has prompted hackers to cheat in order to gain an advantage.

Hackers have used input fabrication attacks successfully against online games. In first-person shooter games, input fabrication attacks are used to inject mouse movements that provide perfect (automatic) aiming and firing. A further use of input injection is weapon recoil suppression – each time a weapon is fired a vector equal to the weapon's in-game recoil is injected into the input stream. In massively-multiplayer games, input fabrication has been used to automatically gather resources. For example, the Glider and Fish-bot cheats send keyboard input to the World of Warcraft game instructing the character to repeatedly fish, mine gold, kill monsters, etc [10][27].

2.1.2 Click Fraud

In pay-per-click advertising, a retailer submits a bid to an advertising agency indicating how much they will pay for a click-through to the retailer's URL. The advertising agent then places a link or alternate form of ad leading to the winning retailer's site, and charges the retailer each time someone clicks on the link.

Pay-per-click fraud comes in several forms [13][25]: *(i)* a bot (or human) can repeatedly click on a competitor's link to deplete the competitor's ad budget, or *(ii)* a contracted agent can repeatedly click on their targeted advertisements in order to generate revenue through a percentage of the bid. As the recognition of click fraud has grown another attack has arisen: *(iii)* one person can try to get another accused of click fraud.

The cost of click fraud is significant. Bids for ads can reach $30 or more per click [11]. Given the ease of automation for input fabrication or fabricated HTTP requests, it is easy to see how the cost of click fraud is estimated at $1B per year [13].

Existing solutions such as CAPTCHAs do not completely solve the click fraud problem due to the time required to solve them. Solving a CAPTCHA can take an average human 51 seconds [6]. A bigger issue is the fact that viewers faced with a barrier to viewing an ad would simply elect not to view the ad. If a technology were available that could detect input data modification attacks, advertisers could be confident that requests coming from clients using that technology were generated by a human, and not a bot perpetrating click fraud.

2.1.3 Auctions, Time-sensitive Sales, Free Services

For online sales that have a temporal component to their value such as eBay auctions, knowledge that a human is initiating transactions with them is valuable. For example, Ticketmaster puts tickets for sale on a first-come-first-serve basis. With a bot, a scalper can automate the purchase of tickets immediately after they are available and then resell them at a higher price. Online auctions can be attacked with a bot that injects a bid at precisely the last possible moment. The outcome of online polls can be biased, and free email accounts can be used to generate SPAM.

For online auctions, time-sensitive sales, and free services, today such bots need not be input-based since they can simply generate HTTP requests for the free service. However, if a solution capable of detecting input data modification attacks was available, these services could use that to prevent boting by requiring the input to those services be verified as unmodified.

2.2 Related Work

Researchers have proposed solutions for online game cheating. These solutions range from protocol extensions that hide key game information and timing analysis from the clients [2][4][5][8], to tamper-resistant software [12]. These solutions do not effectively address the problem of input data modification.

In addition to the academic research in this area, many game developers have included software based anti-cheat infrastructure in their games [21]. Based on the continued existence of functioning game cheats, it is clear that these solutions are not completely addressing input data cheats.

Methods have also arisen to address click fraud. One documented method of detecting fraudulent clicks is by pruning clicks that come from the same source IP address with certain temporal patterns [7]. Such solutions are inherently domain-specific. For example, pruning input that comes too frequently from the same IP address does not work for online games, where all input from a given client often arrives from the same source IP address. Furthermore, such approaches are unable to distinguish legitimate clicks that happen to have the same timing patterns as programmatically generated clicks, such as might arise when comparison shopping [7]. These methods are also vulnerable to botnets, where requests come from diverse source IP addresses.

CAPTCHAs, are a studied area used to prevent bots from attacking time-sensitive sales and free services [1]. Unfortunately, anti-CAPTCHA software has proven to be remarkably effective [17], and CAPTCHAs are not appropriate for latency-sensitive applications like online games and auctions.

TPMs [24] are used in current platforms to provide some security related capabilities. They are not capable themselves of performing computation. A TPM's services could be leveraged by an input data protection system, but a TPM in and of itself is not capable of providing input data verification.

Biometrics can be used to verify the presence of a human via authentication. However, when the system owner is the likely attacker, they can simply choose to use no authentication at all. Using biometric authentication in conjunction with a method for detecting input data modification attacks may result in a more robust system. Biometrics authentication in isolation, however, is not effective at detecting input data modification attacks.

2.3 General Threat Model

The specific attacks discussed in the preceding sections can be generalized into a model describing the class of input-based security threats. This model defines the sources of the attacks and the full range of vulnerabilities presented.

For a general input data threat model, the most powerful source of attack is the platform owner. The platform owner is typically the individual trying to cheat or perpetrate click fraud. Assuming the platform owner is hostile represents a substantial difference between the input verification problem and many other security problems, and it is a guiding principle for this model.

The range of vulnerabilities for this general threat model can be understood by considering a typical software application input processing pipeline. Modern operating systems provide hooks where input data can be legitimately altered on its way to the software application. For instance, input device drivers such as the N52 [19] driver can generate multiple logical input events for each physical input event. Pressing a single key can generate data corresponding to multiple key presses. As long as the software application allows such transformations, the transformed input should be considered legitimate. However, an attacker that uses a driver to insert keystrokes into the application must be detected as illicit.

An attacker willing to use trampoline functions [15] can perform an attack anywhere in the OS or application. Such transformations must be detected as illegitimate. A final class of threats exists from modified HID hardware, or a HID that is itself capable of modifying input data entirely on its own.

Given these attack sources and range of vulnerabilities, the general threat model can be stated as: *Any illegitimate insertion, modification, or deletion of data coming from a human input device between the time the data is created and the time it is consumed by the intended software application.*

3. METHODOLOGY

This section describes how the input monitoring system works. The components that make up the system are described first, followed by a description of the startup flow. The section concludes with details on the system's steady-state operation.

3.1 System Architecture

The detection system's architectural composition is shown in Figure 1. The elements comprising the system are:

Software Application – An application that is being protected from input data attacks (e.g. a game client or web browser).

Input Device – A human input device generating input data that should be protected from insertion, deletion, and modification attacks (e.g. keyboards, mice, trackballs, N52 [19]).

Input Verification Service (IVS) – A service that compares the input streams coming from the *hardware input duplication filters* and the *software application* in order to detect anomalies.

Hardware Input Duplication Filters – Filters that selectively duplicate input data as it arrives from the HIDs. The filters pass all the input data to the operating system as would occur normally, but also send a copy of the input data stream to the IVS.

Remote Service Provider (RSP) – A computing device separate from the client such as an online game server, or web server that provides a service to which the client has agreed to prove that its input data is coming from a HID. Section 3.3.1 describes the requirements placed on the RSP, and its connection to the client.

Isolated Execution Environment (IEE) – An entity capable of executing firmware in a manner that is independent of the client CPU(s). The IEE must be tamper-resistant. Tamper evidence is insufficient because an attacker could disable reporting mechanisms to the RSP, thereby circumventing tamper-evidence generation. The code run by the IEE must be produced by a trusted entity such as the platform manufacturer. The IEE must have a communication path with the client CPU that is tamper-evident to insertion, modification, or deletion attacks against the data moving across it as discussed in section 3.3. In our prototype the IEE is implemented using the Manageability Engine contained in Intel® chipsets with Active Management Technology [16].

The trust model of this system requires that the platform silicon and board manufacturers, the software running in the IEE, and the remote service provider are trusted. Other system components such as the operating system and window manager do not need to be trusted, and are assumed to be under the control of the attacker.

3.2 System Startup

Once a software application elects to have its input data stream protected with this system, the application performs the steps shown in Figure 2 to start the detection system.

After the software application has started and established the identity of the remote service provider, it must send configuration information to the IVS. The configuration information includes:

1. The input devices the application will accept input from
2. The identity of the remote service provider
3. Input translations allowed by the software application (3.3.2)
4. Application identifying data, e.g. an integrity manifest [23]

The IVS uses the RSP identity for reporting detected attacks. The input device list is used to program the input duplication filters. For instance, an application may allow input from a USB keyboard and mouse, but not a PS/2 trackball. The IVS must perform run-time integrity verification of the software application. This is done using application identifying data. Integrity verification is required to prevent the detection system from being enabled, then rendered ineffective through modifications to the software application as discussed in 4.3.

An initial synchronization problem exists with absolute HID state such as the absolute mouse location. This is addressed by using the initial input events to synchronize the IVS with the software application's view of the input devices' states.

3.3 Steady State Operation

In steady state operation, verification is performed on the input data streams from the input duplication filters and the software application. The sequence of steps performed by the IVS during steady-state operation is shown in Figure 3.

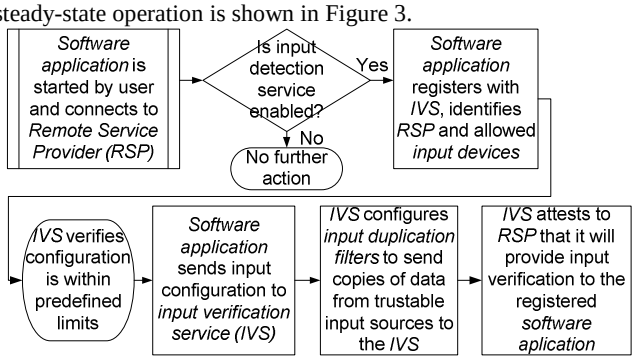

Figure 2: System Startup Flow

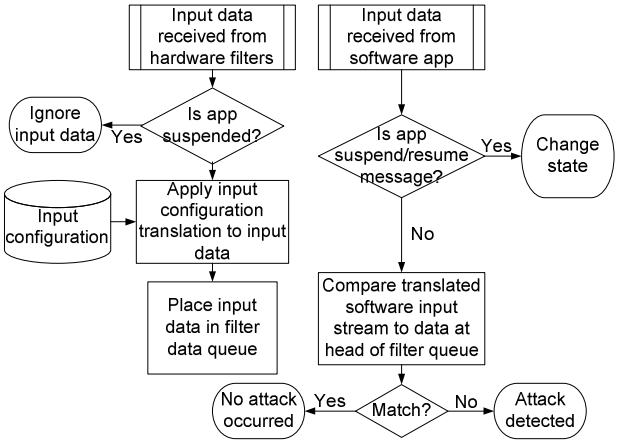

Figure 3: Input verification service steady-state operation

The IVS takes each input event received from the input duplication filter, translates it (see section 3.3.2), and places the resulting data on an event queue. Example input events are a key being depressed, a mouse movement, etc. The IVS then compares the input event(s) on the filter queue to those on the software application queue. If the events match no tampering has occurred and the matching events are discarded from both queues. If they don't match, an attack is underway.

When an input data attack is detected, the IVS notifies the RSP over a reliable, tamper-resistant connection. This connection must provide message integrity, message duplication detection, data origin authenticity, and message confidentiality (confidentiality is required for online gaming, but not click fraud). The TLS [9] and DTLS [22] protocols meet these requirements. When establishing this connection, the IVS presents credentials signed by a manufacturer that are unique from, and inaccessible to, the rest of the client system. Session keys are stored in the private memory of the IVS, so they are inaccessible from the client computer. Storage and use of these credentials and session keys compel the requirement that the IEE be tamper resistant.

The communication path between the software application and IEE must be protected from attacks against message data. This is because an attacker could change input events before they arrive at the software application, and then modify them again during their transit to the IVS. This type of attack can be mitigated by performing an integrity check on the host driver communicating with the IVS, and attaching a message authentication code to each message that is verified by the IVS, as described in [23].

In order to ensure user privacy is preserved, the IVS sends no input data to the RSP – only an alert indicating that a discrepancy in input data was detected. In addition, the IEE can only run code signed by a trusted entity such as the platform manufacturer.

3.3.1 Remote Service Provider

The RSP must verify it is communicating with a real IVS by authenticating the IVS' credentials. These credentials should be unique per platform and provisioned by a trusted entity. This avoids privacy issues but requires a group signing mechanism. In order to ensure a man-in-the-middle does not drop attack detection notifications, the RSP must verify no messages are lost.

In certain applications like online gaming, the RSP itself is not necessarily a trusted entity. The owner of the RSP may be the attacker. In order to address this scenario, the software application on the RSP can be required to go through the same verification process that the client software application is subject to. This includes run-time integrity verification of executable code, and authentication of the RSP's IEE credentials.

3.3.2 Input Translations

Some input devices perform software translations on input events in a way that may be considered legitimate by certain applications. For example, devices like the Nostromo N52 [19] have drivers that translate one button press into multiple input events. For example, a single key press could be translated into three keystroke events. The IVS must treat translated input as legitimate when the software application allows it.

To allow legitimate input macros from monitored HIDs, the IVS requires the software application to identify the translations to be used for each input device, and send them to the IVS. Once the IVS has the translation rules, it applies them to the input data received from the HIDs. Since these same translations were applied in the same way by the application, this translated data will match the translated data processed by the application exactly - provided no attack is underway.

3.3.3 Application Suspend/Resume

Application suspend occurs when a protected software application is relegated to the background and input data is sent to another application. Many different events can cause application suspend and resume, so the protected software application must notify the IVS when such an event has occurred. Since the application and its communication path with the IVS is integrity checked an attacker cannot tamper with this notification.

4. EVALUATION

The effectiveness of a system for input data attack detection is determined by its performance overhead, implementation cost, detection effectiveness, and cost of circumvention. A prototype of this design was implemented and evaluated against these criteria. The prototype was shown to detect input attacks against the online game *Quake 3*. This section describes the prototype, how it performs against the evaluation criteria, as well as some assumptions and limitations of the design.

4.1 Prototype

The prototype was constructed on an Intel® x86 platform. The software application and RSP were Quake 3. The IEE was the Intel® Manageability Engine (ME), a microcontroller present in the platform's chipset [16]. The IVS was run on the ME, and the HID filter was a USB driver. This HID filter implementation deviates from the system architecture described in Section 3.1, and was used only to reduce the hardware development cost of the prototype. This implementation is vulnerable to circumvention. A real implementation must place the filters in hardware.

A diagram of the prototype is shown in Figure 4 with dashed lines depicting deviations between the prototype and the general system architecture shown in Figure 1. Modifying Quake 3 to work with the user library required about 600 additional lines of code. Once the RSP receives a cheat notification from a client IVS, it notifies all connected clients of the cheating. When clients receive the cheating indication, a message is displayed on the screen stating who has been caught cheating. Other remediation is possible, such as dropping cheaters from the game entirely, or banning them from the service.

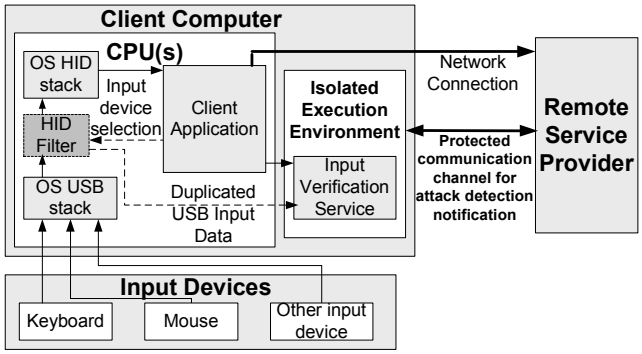

Figure 4: Prototype components

4.2 Performance Analysis

Game engine performance is often measured in frames per second (fps), but human driven benchmarks produce large fluctuations in fps since changes in the point of view of the player induce different computational loads from one run to the next. Thus, this performance analysis uses relative clock cycles averaged across multiple runs to measure performance impact.

The computational overhead of collecting user input and forwarding it on to the IVS is shown in Table 1, which quantifies the total number of cycles used during a period of game play as well as the number of cycles used for the verified input calls.

Table 1: Prototype performance measurements

Total Game Time		Verified Input Added Time	
Frames	Cycles	Messages Sent	Cycles
107281	537,386,310,090	14084	1,392,274,809

The measured data shows that sending the user input messages to the IVS consumes less than 0.25% of the total cycles used by the client application. With optimization, the client application overhead can be reduced below this already small impact.

4.3 Assumptions

Two important assumptions are built into the detection system described in this paper. First, this solution assumes the software application's integrity is maintained. This includes its executable code, control flow, and dynamic data. Well-known mechanisms exist for maintaining software integrity [3][12][20][23], but they have limitations in terms of performance and completeness. The detection system described would be subject to the same limitations and vulnerabilities present in any system used for integrity verification of the software application.

The second assumption is the communication path to the IEE must be verified. An attack against the communication path could remove an input event from the data stream before it arrives at the application and then insert the same event into the message queue going to the IVS. This would make it appear as though an input event was processed by the application when in fact it was not.

4.4 Detection Effectiveness

This input monitoring method detects real input data attacks that are in use today. It is capable of detecting the attacks described in sections 2.1 and 2.3; exceptions are described below. The prototype demonstrated the system's effectiveness against input injection attacks and detected an aimbot that automatically aimed a user's weapon in Quake 3. It was also tested and proven effective at detecting the use of trampoline functions.

4.5 Unmitigated Threats and Limitations

There are a number of attacks that are not mitigated by the input protection service described here. These include:

1. A HID containing hardware that stores macros executed as a result of a single button press such as the MCK-142 [18].
2. Platform hardware modifications
3. A robotic key pressing device.
4. Use of an in-circuit emulator or bus analyzer
5. Attacks that alter the timing (but not order) of the arrival of input data at the application.
6. Attacks against the initial position of the mouse

Attack 1 is interesting due to its modest cost to wage. This input detection system by itself cannot detect these attacks, it would need to be used in conjunction with a 'trusted HID' – one that can attest to its correct, non-macroed operation through the use of a manufacturer embedded secret – in order to detect thesm.

The input protection service also does not protect against any hardware modification attacks. Soldering chips on or off the board are example hardware modification attacks. It is currently unnecessary for attackers to wage hardware attacks on PCs, since software attacks can achieve the same results at lower cost.

It is possible to alter the time at which input data arrives at the application for processing in an undetectable manner. Alterations to ordering, or additions/deletions of data would however be detected, so this attack is of limited potential.

The system is vulnerable to attacks that alter the initial absolute position of the mouse when the system is enabled. Relative mouse movements subsequent to the initial position can be reliably tracked, however if something can be accomplished by altering the initial absolute mouse position, such an attack would not be detected by this system.

This detection system also faces some real-world deployment limitations. USB is currently the prevalent protocol for input devices, but other protocols exist such as PS/2 and Bluetooth. The input duplication filters must filter data coming from all bus types that can be trustable sources. An ideal place within the platform where an IVS would have this capability would be in the I/O controller hub. Supporting multiple protocols though would increase the implementation cost of the system.

A further limitation is the system's complexity. In practice complexity leads to implementation induced security vulnerabilities. A real deployment of this system would need to balance complexity-based risk and fullness of functionality.

4.6 Cost

The deployment cost of this input detection system is relatively low. The prototype has been implemented using hardware that already exists on many PCs. One additional piece of hardware would be required (the *Hardware Input Duplication Filters*), as well as some software components. The moderate cost required to wage an attack against this system via on-device programmable HIDs means this solution would be most effective for moderate value security applications such as online gaming and click fraud. It provides a reasonable barrier to circumvention as measured by the cost of hardware modification attacks commercially available for the purposes of cheating on gaming consoles. High value applications will require this system to be used with additional protection mechanisms in order to raise the cost of circumvention.

5. FUTURE WORK

Despite the progress represented by the input protection service described in this paper, additional work in this area remains. The proposed input monitoring method goes well beyond existing capabilities in proving input data has come from a physical human input device, but the fundamental question of 'is a bot at the controls?' remains unanswered. Biometric authentication devices designed to make their data stream hard to forge programmatically could be used in conjunction with this input protection service. This may help answer the question of human presence and is one vector that warrants investigation.

Given a tamper-resistant isolated execution environment, the hardware architecture of input devices should be re-examined. Such an environment could provide a robust way of authenticating certifiable input hardware devices. This could allow for an end-to-end, high integrity input path.

Protecting against hardware attacks is another area that warrants additional investigation. While it will never be possible to defeat all hardware based attacks, it may be possible to significantly raise the bar against these types of attacks.

6. CONCLUSIONS

This paper describes a method that is capable of detecting a class of attacks that modify input data coming from a human input device. By isolating an input data verification service from the platform owner, this method provides a tamper-resistant way of detecting attacks against human input data. The prototype constructed demonstrates the effectiveness of the architecture at detecting real-world attacks in use today. Experimental results show that the method induced negligible performance impact on a demanding application. In addition, the development cost of integrating the method with new applications is small. Based on this data, the input data monitoring method described here can be used to effectively protect applications from input data attacks such as click fraud and cheating in online games.

7. ACKNOWLEDGMENTS

The authors would like to thank Moshe Maor for contributing ideas that went into the design and architecture of this system.

8. REFERENCES

[1] L. von Ahn, M. Blum, N.J. Hopper, J. Langford. "CAPTCHA: Telling humans and computers apart." *Advances in Cryptology, Eurocrypt '03*, volume 2656 of *Lecture Notes in Computer Science*, 2003, 294–311.

[2] N. Baughman, and B. Levine. "Cheat-proof Playout for Centralized and Distributed Online Games", *IEEE INFOCOMM*, 2001.

[3] L. Catuogno, I. Visconti. A Format-Independent Architecture for Run-Time Integrity Checking of Executable Code. *Proceedings of the Third International Conference on Security in Communications Networks*, 2002.

[4] C. Chambers, W. Feng, W. Feng, Saha, D. "Mitigating Information Exposure to Cheaters In Real-Time Strategy Games." *NOSSDAV'05*, June 2005, Stevenson, Washington.

[5] B. Chen, M. Maheswaran. "A Cheat Controlled Protocol for Centralized Online Multiplayer Games." *SIGCOMM'04 Workshop on NetGames*, September 2004.

[6] M. Chew, J.D. Tygar. Image Recognition CAPTCHAs. *Proceedings of the 7th International Information Security Conference*, September 2004.

[7] Click Quality Team. "How Fictitious Clicks Occur in Third-Party Click Fraud Audit Reports" *Google, Inc.* August 8, 2006. http://www.google.com/adwords/ReportonThird-PartyClickFraudAuditing.pdf

[8] E. Cronin, B. Filstrup, S. Jamin. "Cheat-Proofing Dead Reckoned Multiplayer Games (Extended Abstract)", *Proceedings of ADCOG*, January 2003.

[9] T. Dierks, E. Rescorla, The TLS Protocol Version 1.2. *RFC 4346*, October 2006.

[10] Fish-bot for World of Warcraft. http://www.fish-bot.com/

[11] E. Feldblum, "Stopping Click Fraud: From One Victim To Another", Pay Per Click Universe, (no date). http://www.payperclickuniverse.com/pay-per-click-search-engines-articles.php?article_id=43

[12] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection based Method for Intrusion Detection. *Network and Distributed Systems Security Symposium Conference Proceedings*, 2003.

[13] B. Grow, B. Elgin, M. Herbst. "Click Fraud: The dark side of online advertising." *Business Week Online*, October 2, 2006. http://www.businessweek.com/magazine/content/06_40/b4003001.htm

[14] G. Hoglund, J. Butler. *Rootkits*, 2006.

[15] G. Hunt, B. Brubacher. "Detours: Binary Interception of Win32 Functions." *Proceedings of the 3rd USENIX Windows NT Symposium*, pp. 135-143. Seattle, WA, 1999.

[16] Intel® Active Management Technology. http://www.intel.com/technology/manage/iamt

[17] May, M. "Inaccessibility of CAPTCHA: Alternatives to Visual Turing Tests on the Web." *W3C Working Group Note* 23 November 2005. http://www.w3.org/TR/2005/NOTE-turingtest-20051123/

[18] MCK-142 Programmable Keyboard. http://www.monu-cad.com/keyboard.htm

[19] Nostromo™ SpeedPad N52. A human input device designed for gaming applications. http://catalog.belkin.com/IWCatProductPage.process?Product_Id=157024

[20] N. Petroni, T. Fraser, J. Molina, W. Arbaugh. Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor. *Proceedings of the 13th USENIX Security Symposium*, 2004.

[21] Punkbuster Online Countermeasures – a software based anti-cheat solution. http://www.evenbalance.com

[22] Rescorla, E., Datagram Transport Layer Security. *RFC 4347*, April 2006.

[23] T. Schluessler, et. al., "Runtime Integrity and Presence Verification for Software Agents", *Technology@Intel Magazine*, December 2005.

[24] *TCG TPM Specification version 1.2 Revision 94*, 2006.

[25] Tuzhilin, Alexander. "The Lane's Gift v. Google Report." http://googleblog.blogspot.com/pdf/Tuzhilin_Report.pdf

[26] Virtual Economy Research Network: News, research and discussion on real-money trade of virtual property globally. http://virtual-economy.org/

[27] Glider - http://www.wowglider.com/FAQ.aspx