INTEGRIKEY: End-to-End Integrity Protection of User Input

Aritra Dhar
ETH Zürich
aritra.dhar@inf.ethz.ch

Der-Yeuan Yu
ETH Zürich
dyu@inf.ethz.ch

Srdjan Čapkun
ETH Zürich
srdjan.capkun@inf.ethz.ch

***Abstract*—Networked critical systems, such as Programmable Logic Controllers in a factory plant, are often remotely configurable by administrators through web-based interfaces. However, administrative host machines have been compromised in recent incidents, allowing attackers to covertly alter user commands or configurations to disrupt the proper function of remote controllers. While most existing approaches focus on securing field devices from malicious programs, the integrity of configuration commands remains to be explored.**

In this paper, we consider the presence of an untrusted host machine and aim to ensure the integrity of user input to a web server directly from a peripheral, such as a keyboard. We propose INTEGRIKEY, an end-to-end integrity protection system that leverages a user-side trusted device (the INTEGRIKEY bridge) and a small server-side software component to ensure the integrity of the user's input. Based on our solution, we also identify a new form of attack, the (user interface) UI input integrity manipulation attack, where a compromised host alters the UI to mislead the user into entering incorrect data. We provide a comprehensive analysis of these attacks and the corresponding solutions. INTEGRIKEY allows the server to accept only authentic user input even when the attacker compromises both the host machines and the network. INTEGRIKEY requires no additional software on the user's host and does not significantly affect the way the user interacts with the system. We implement INTEGRIKEY in the context of remotely configuring Programmable Logic Controllers and our evaluation shows that it incurs minimal overhead in securing user input integrity.

I. INTRODUCTION

Nowadays, remote configuration and reprogramming, and more generally, remote user input into a networked device are commonly performed through web-based technologies. Typically, a configurable device (e.g., a Programmable Logic Controller, a medical device, a home automation system or an IoT device) runs a web server that is accessed by a user's host, such as a computer in the control center of a plant. The web interface allows users to configure the device or provide some other relevant input from their hosts. Today, a wide range of embedded systems can be configured in such a way [1]–[5]. If the device does not run a web server, its remote control may still be exposed through some dedicated web servers. Thus, enabling user control from any host that runs a web browser.

Unfortunately, users' hosts and the control network used to configure critical (embedded) systems are often targets of attacks, such as exploiting zero-day vulnerabilities [6] and social engineering [7]. One example of such attacks is Stuxnet, where the compromise of the user's host in the control center led to the misconfiguration of PLC programs that damaged a power plant [8]. While web servers in critical systems can use Transport Layer Security [9] to protect their communication from network-level attackers, they are not secured against host compromise or man-in-the-browser attacks. In particular, compromised hosts can alter the commands that are issued by users to covertly misconfigure remote devices and cause damage. Although this problem can be partially addressed using system hardening and best security practices, the complexity of modern critical infrastructures is often too high to systematically prevent malicious inputs or user negligence.

In this paper, we focus on the preserving the integrity of users' input. We propose INTEGRIKEY, an integrity verification system that protects user's input from an input peripheral, such as the keyboard, to a configurable end device. Our system consists of two components: *(i)* the dedicated INTEGRIKEY bridge, which serves as a bridge between the input peripheral and the user's host, and *(ii)* a lightweight and trusted component running on the end device that verifies the integrity of user commands. INTEGRIKEY does not require any trust in the user's host computer, the browser used to access the web interface, or any other (e.g., cloud) component that connects the host to the end device.

INTEGRIKEY works by connecting input peripherals, such as a USB keyboard, to the user's host via the INTEGRIKEY bridge. This bridge can therefore directly record the user's raw input and report it to the end device via the web server in an authenticated manner. To support this, we use the new WebUSB browser API, which allows websites to communicate with USB devices connected to the host. Using INTEGRIKEY, the end device receives the user's input from both the untrusted host browser (which sends user's inputs into the web form) and the INTEGRIKEY bridge. The end device verifies the input that it receives from the host against the data received from the INTEGRIKEY bridge while requiring only minimal changes in the user's behavior. The user can still use the input peripheral, browser, and web interface in a similar manner as if INTEGRIKEY was not deployed. INTEGRIKEY, therefore, preserves system usability while introducing a small trusted component at both the host's and the device's side to achieve end-to-end input integrity.

However, using a dedicated secure channel to ensure the integrity of user input is not a trivial task when the entire host is compromised. This scenario gives rise to a new UI

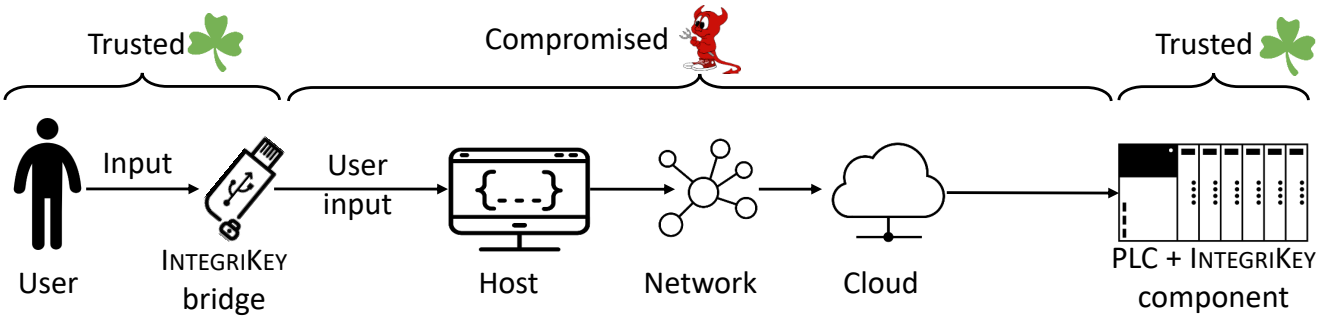

Fig. 1: **We assume two possible scenarios: (i) end-system (e.g., PLC) runs its web server and (ii) end-system is placed behind a cloud service that runs the web server and forwards the traffic to the end-system. The user accesses the end-system through the browser in her host. We assume that the user and the end-system are trusted, whereas the host, the network, and the cloud are untrusted and fully controlled by the attacker. We further assume that the user can deploy a trusted device (e.g., USB key) and runs a browser that supports WebUSB/WebBluetooth (Google Chrome currently supports these APIs).**

input integrity manipulation attack that is so far not sufficiently addressed in the research. Recent work on UI manipulation attacks mainly explored the context of authenticity and user privacy [10], [11]. More specifically, although one can provide end-to-end integrity guarantees on user input, the attacker can still manipulate the appearance of the web page to influence the sanity of the user's input. For example, the attacker can change the labels of a certain field to trick the user into entering a wrong configuration value. Similarly, the attacker can reorder the labels of fields to trick the user into entering values in the wrong order. Such attacks affect the normal function of the device if the possible values in the reordered data fields are still considered valid input by the device. Based on users' different levels of familiarity with the web page, we explore the possible web page manipulation attacks and propose a comprehensive solution to reassure the integrity of the user's input. In particular, we develop the INTEGRIKEY tool to identify data fields that are susceptible to label reordering attacks. We then modify the server to instruct users to manually add the label to the input value for fields whose values may be swapped with other fields.

To demonstrate the feasibility of this approach, we develop a prototype implementation consisting of the INTEGRIKEY tool and the INTEGRIKEY bridge using an Arduino board. We use the tool to process typical data fields in a wide range of applications supported by x600m, a commercially available browser-based PLC server. We also evaluate the runtime performance of the bridge, and our results show that the bridge adds no more than 1 second to the loading of the web page on the browser and only $\sim$ 50 milliseconds to the delay on user input from the user side.

The contributions of this paper are as follows:

- We propose INTEGRIKEY, a system that preserves the end-to-end integrity and authenticity of user input to a web server in the presence of a compromised host.
- We identify a new type of an attack: the UI input integrity manipulation. We analyze this attack, by which the attacker can trick the user into entering values in the wrong fields. We further design a set of measures that prevent such attacks and integrate them into INTEGRIKEY.
- We design and develop a server-side INTEGRIKEY tool that analyses a configuration webpage to determine the set of input fields that are vulnerable to the UI input integrity manipulation attack.
- We show the feasibility of our approach by developing a proof-of-concept implementation. Our evaluation shows that INTEGRIKEY adds only minimal overhead to the user's operation.

The rest of the paper is organized as follows. In Section II, we introduce the problem of preserving the integrity of users' input into a remote web server. Section III describes the system design of INTEGRIKEY. In Section IV, we describe the UI input integrity manipulation attack and present our comprehensive solution. Section V and VI provides detailed description of the INTEGRIKEY's prototype implementation and evaluation of the prototype, respectively. In Section VII, we discuss various aspects of INTEGRIKEY. We summarize related work in Section VIII and conclude in Section IX.

II. PROBLEM STATEMENT

The main problem that we are trying to solve in this work is to protect the integrity of end user's input into a remote web server that runs on a critical end device (e.g., PLC). We focus on scenarios where the user is a remote operator whose task is to configure such end devices. We illustrate this scenario on Figure 1.

Protecting user input integrity is relatively simple when the client's system is trusted and properly configured: a TLS session from the user's browser to the server guarantees not only the integrity but also the confidentiality of all data exchanged between the client and the server.

However, numerous revelations have shown that sophisticated attackers are capable of compromising the users' host systems, such as personal computers or smartphones, to gain control over the communication channel and alter the users' input to critical infrastructures. Such attacks can be orchestrated by compromising the operating system, planting hardware bugs in the system (such as the production line attacks [12],

[13]), installing malware, etc. For the user's input, there are also simple attacks such as a hardware or software-based key-loggers that intercept the input from the keyboard or the mouse and modifies them. For web-based control interfaces, browsers are complex applications and often susceptible to zero-day attacks or exploits [6], [14], [15]. The wide range of attacks on the host system grants the adversary many ways to intercept a user's input to remote web servers.

Motivated by the large attack surface of a typical user's host system, we aim to ensure the integrity of the user's input data entered into a web-based interface of a remote server. We assume that the user, the user's input peripheral (e.g., keyboard), and the remote server are trusted.[1] However, we assume that the adversary compromises the user's host system, potentially by some aforementioned attack, and is a Dolev-Yao attacker that can eavesdrop, delete, and inject messages into the communication channel between the host and the remote server.

In addition to our goal, we also identify some design constraints for the ease of adopting our solution. Such constraints in our solution to this problem should not change the user's interaction with the host. She should be able to interact with the end-system via the browser running on her host in a manner to which she is used. Furthermore, there should be no need to install or maintain any new components on the host. We also not assume any other communication channel except the one from the host to the web server.

III. INTEGRIKEY: SYSTEM DESIGN

In what follows we describe the design of INTEGRIKEY in more detail.

A. System Assumptions

We assume a conventional client-server system which primarily involves three components. The USB/Bluetooth enabled input peripheral (keyboard, mouse, bulk storage), a host which is a computer/smartphone, that connects to the remote server through a web browser and the remote server where the host is connected. In addition, we assume that the user's browser supports WebUSB/WebBluetooth. WebUSB is an upcoming standard in development [16] by Google and is currently implemented as a feature in Google Chrome. WebUSB allows JavaScript code served from a HTTPS secured page to have direct access to USB devices. The WebUSB API safely exposes USB functionality to the JavaScript code allowing the website to act as a USB host and communicate with the USB devices.

B. INTEGRIKEY

INTEGRIKEY aims to protect the integrity of user's input in highly adversarial settings, where user's host and all intermediary systems that transfer and process user's input to the server are compromised. Furthermore, INTEGRIKEY aims to achieve this without changing user's existing behavior and without requiring additional software to be installed on user's host system, therefore, facilitating deployment.

INTEGRIKEY achieves this by adding two components to the common client-server: an embedded INTEGRIKEY bridge, which captures the user's input (keyboard/mouse) and interacts with the user's browser and therefore with the server via WebUSB, and a INTEGRIKEY integrity verification server component that detects integrity violations of user's input.

As shown in Figure 2, we assume that the user connects its input peripheral to the INTEGRIKEY bridge, and further connects the INTEGRIKEY bridge to its host system. This way, the INTEGRIKEY bridge acts like a bridge between the input peripheral and the host. These connections can be wired or wireless depending on the interfaces that the host and peripheral expose which will be different in the case of a phone and computer.

When the user launches her browser and starts a connection to the server, the server will be able to reach the INTEGRIKEY bridge via WebUSB. The server will, then create two connections, one to the browser (to serve content and take input from the user interface), and the other one to the INTEGRIKEY bridge (through the browser via WebUSB). The connection to the browser will be protected using HTTPS (TLS). We assume that common user authentication is used. The integrity (and confidentiality) of the communication between the server and the INTEGRIKEY bridge will also be protected. This can be achieved in a number of ways, e.g., via pre-shared keys, or via a fresh TLS session. We chose the latter. In INTEGRIKEY, the server and the INTEGRIKEY bridge exchange public key certificates and establish a TLS channel that will protect their communication. This does not require a setup of a new PKI, INTEGRIKEY can also leverage the existing PKI that is used in today's web. The INTEGRIKEY bridge can be periodically updated with relevant CA root certificates.

When the user enters her input (text, mouse clicks), her input will pass via the INTEGRIKEY bridge. The INTEGRIKEY bridge will (i) forward the input to the browser so that it is shown on user's screen and sent via HTTPS to the server and (ii) record the input and send it to the server via INTEGRIKEY bridge-server TLS connection. The server will then compare what it received from the browser and from the INTEGRIKEY bridge. If the input values received from the INTEGRIKEY bridge correspond to the values received from the browser, the server will conclude that the input from the browser is legitimate. The server will then indicate to the user (e.g., via an LED indicator on the INTEGRIKEY bridge) if the input has been successfully 'committed' to the server. This last step is optional and depends on whether such assurance is critical for the application.

In INTEGRIKEY the user's experience is largely unchanged. She will interact with the server through the web browser, fill in the forms and submit them to the server in the same manner as in any other system.

The main steps of the execution of INTEGRIKEY are shown on Figure 2.

1) The user starts a browser and types in the URL of the

[1] If the remote server is compromised, then no countermeasure would work since it can directly actuate in a malicious way.

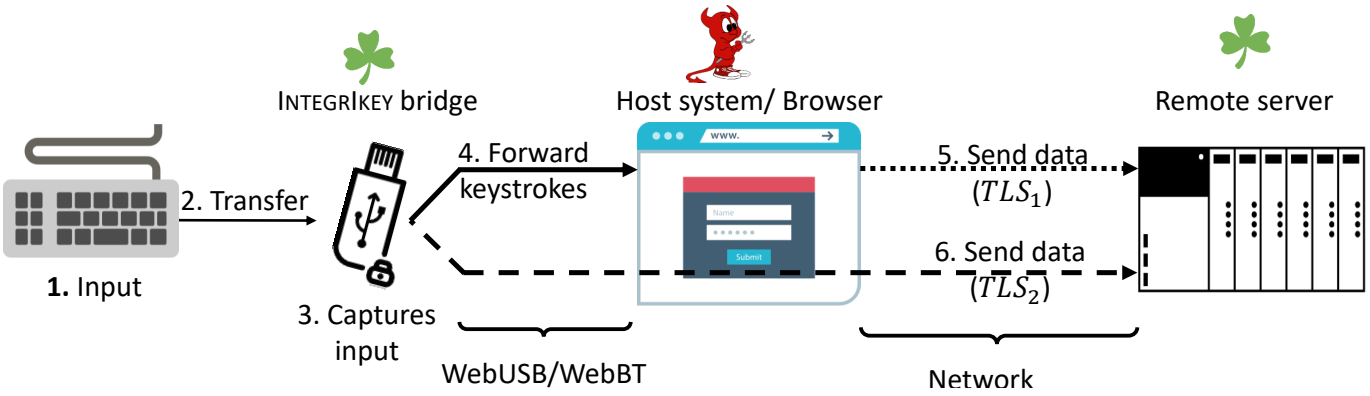

Fig. 2: **INTEGRIKEY operation. The user starts the browser and types in the url of the web server. The server will serve the page that contains a script that will instruct the browser (via WebUSB) to contact the INTEGRIKEY bridge. Two secure connections will be created: an HTTPS (TLS_1) connection between the Browser and the Server and a secure connection between the INTEGRIKEY bridge and the Server (TLS_2). When the user provides the keyboard input, the INTEGRIKEY bridge will send this input to the browser (to fill in the web form), and also send it directly via the TLS_2 connection to the server[2]. Upon receiving the data from both connections, the server compares them and accepts the input from the browser if it matches the one from the INTEGRIKEY bridge.**

web server. The server serves a script to the browser that invokes WebUSB API, establishing a secure communication channel to the INTEGRIKEY bridge.

2) The user enters her input into the input peripheral device that is connected to the INTEGRIKEY bridge. (steps $1-2$).
3) The INTEGRIKEY bridge captures the keystrokes from the input peripheral (step 3).
4) The INTEGRIKEY bridge forwards the input to the browser (step 4). When the user submits the form, the browser sends the input data to the remote server via the HTTPS (TLS_1) channel (step 5).
5) The INTEGRIKEY bridge further sends the keystrokes directly to the server through the secure channel (TLS_2) between the INTEGRIKEY bridge and the server (step 6).
6) Upon receiving the data from TLS_1 and TLS_2, the server compares the data. If the keystroke data from TLS_1 matches with the data from TLS_2, the server accepts them and sends a feedback signal to the INTEGRIKEY bridge via TLS_2.
7) The INTEGRIKEY bridge shows the feedback to the user to notify that the proper input is recorded by the remote server. This feedback could be displayed via an attached LCD screen or a LED light.

C. Security Analysis

We assume a strong attacker that controls host system (that includes the hardware and the operating system), the browser, the cloud/network and any other intermediate component on the path to the server. As the attacker controls the host system, it can manipulate or create fake user input from the browser to the server (TLS_1), as well as control the user interface.

[2]TLS connection can be replaced with any authenticated connection, based on digital signatures or shared keys between the INTEGRIKEY bridge and the server. For efficiency reasons, it is also possible that the TLS_2 only carries the signature of the keystrokes and not the data itself.

We assume that the remote server is a trusted entity and has a public key certificate and corresponding public-private key pair to prove its authenticity by leveraging an existing PKI. The server serves a JavaScript snippet that uses WebUSB API to communicate with the INTEGRIKEY bridge. The INTEGRIKEY bridge has a public/private key-pair and a public key certificate that is used to establish a secure channel (TLS_2) with the server. The server can validate such certificates e.g., through a company PKI. The user attaches all her peripheral devices through the INTEGRIKEY bridge to eliminate any input manipulation or forged data generated by the malicious host. We assume an authenticated TLS connection setup. Given this, the host cannot impersonate the INTEGRIKEY bridge or inject input into the TLS_2 channel that was not generated by the user. Additionally, the INTEGRIKEY bridge also provides a feedback to the user indicating if it has been successfully reprogrammed. In case the INTEGRIKEY bridge fails to provide the feedback, the user concludes that the attacker is executing a denial of service attack. Protection against denial of service is outside the scope of this work.

Since the attacker cannot inject messages into the TLS_2 channel, the only remaining possibility for the attacker to manipulate the input is to change the user interface (which the attacker controls by controlling the host) such that the user inputs values into the incorrect fields. This attack is feasible since the INTEGRIKEY bridge only registers the input keystrokes and their sequence, and assumes in which sequence they need to be filled. However, the INTEGRIKEY bridge and the server do not know which interface user is currently seeing on the screen. E.g., the fields can be reordered by the adversary. Depending on the end-system and its function, it might happen that these user interface attacks prove detrimental.

We explore this attack further and propose countermeasures in Section IV.

D. Alternative Channels

In our design of INTEGRIKEY, the dedicated connection that the INTEGRIKEY bridge establishes with the server shares the same physical channel as the browser, i.e., the internet connectivity of the host. However, INTEGRIKEY can be configured in such a way that the communication channel of the INTEGRIKEY bridge remains separated physically from the host. This can be achieved by using a smartphone application as the INTEGRIKEY bridge. The user connects her peripheral devices with the smartphone, the INTEGRIKEY bridge application communicates with the USB peripheral device and interprets user inputs. Then the application forms a separate TLS channel using its own network (using WiFi or cellular data). This setup does not require any dedicated hardware device. However, it requires trust in the smartphone and additional applications installed in it. Moreover, such setup may not be suited for the industrial PLC operators.

IV. PROTECTING AGAINST UI INPUT INTEGRITY MANIPULATION ATTACKS

In the previous section, we described how INTEGRIKEY bridge defends against the attacker's manipulation of input values. In this section, we describe and propose a solution for more sophisticated attacks that compromise the integrity of users' input by manipulating graphical elements on the browser. Our attacker model remains identical to the previous solution, i.e., an attacker that compromises entire host system and the network communications. The malicious host can employ manipulation on the graphical interface elements such as adding extra fields or removing some of the fields. This type of data is effective against inexperienced users who are not properly familiar with the user interface and the specific input fields in a page. Note that when the server serves a specific website, it knows what to expect when the user submits the data into the browser. Therefore, these attacks can be readily detected by the server using the former method due to the discrepancy in the number of input data.

If the user is experienced with the user interface, a malicious host can orchestrate a more sophisticated attack that swaps the labels of input fields on the served web page. Here, the attacker does not manipulate input values (since these values will be signed by the INTEGRIKEY bridge) but instead presents false labels in the web form to the user. The user is then tricked into entering the data into incorrect fields.

The experienced users who are familiar not only with the user interface, but also know which data fields to expect are not susceptible to such kind of attacks. However, a malicious host can always show a notification falsely claiming that the server updated the UI elements due to a version upgrade. This way the malicious host can trick even a very experienced user into putting data into a wrong field. Therefore, we can conclude that even the most experienced users are susceptible to the UI manipulation attack that compromises users' integrity.

One can argue that serving a manual or displaying the template (as security indicator) of the website in a separate display (may be attached with the INTEGRIKEY bridge) can prevent UI manipulation attack. However, several existing pieces of research [17], [18] show that in practice users tend to ignore security indicators. Therefore, one of the design goals of INTEGRIKEY was not to rely on any security indicator that increases user's cognitive load. INTEGRIKEY rather asks the user to perform a compulsory but a very simple task that associate her input data with the field label which she is seeing on the host's screen. We argue that the additional task that the user performs is extremely simple in nature and does not change the natural user behavior in any significant way.

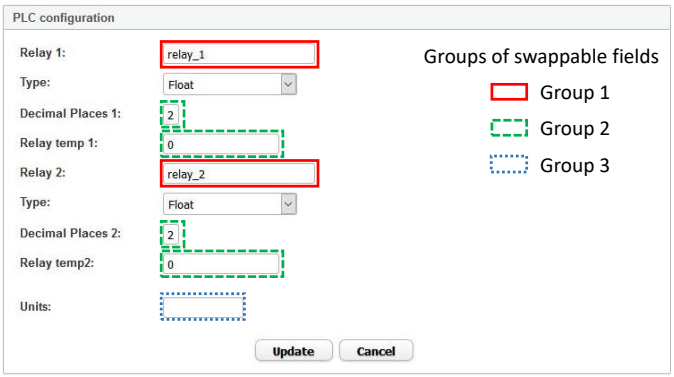

Fig. 3: **An example of a web-based PLC configuration page where the highlighted fields can be reordered by a malicious host. Note that Relay 1 and Relay 2 are interchangeable, where as Decimal places 1, 2 and Relay temp 1, 2 all are swappable.**

THE ATTACK. Field data types can easily be identical but interchanging the values may change the behavior of the end-system. The server cannot notice this change since the signed values from the INTEGRIKEY bridge will be no different from the values entered through the web form and could be plausible values and entered in a plausible sequence for the served web form. E.g., the attacker can swap the labels of fields 'Decimal Places' and 'Relay temp 1' in the web-form illustrated in Figure 3. The user will then enter the values in a sequence ('Relay temp 1', 'Decimal Places'). The server will interpret this as a sequence ('Decimal Places', 'Relay temp 1') since this is the order that should have been imposed by the web form. Since both values are in the expected ranges, the server will not notice the swap, unless these values need to be related (e.g., 'Decimal Places'<'Relay temp 1') and the server checks for this relation. However, in many systems, one cannot rely on the end-device logic to check for such consistency and there are many values that will overlap in range but will otherwise be fully independent. Such protections, therefore, do not easily generalize.

To prevent this attack, we, therefore, propose a mechanism which binds input data to the input label. There are two parts to this proposed solution, one of which is executed on the server and the other on the INTEGRIKEY bridge at the user side.

A. Server Side Mechanisms

We develop a tool that parses an existing web page form and extracts the input fields. The tool then calculates the set of

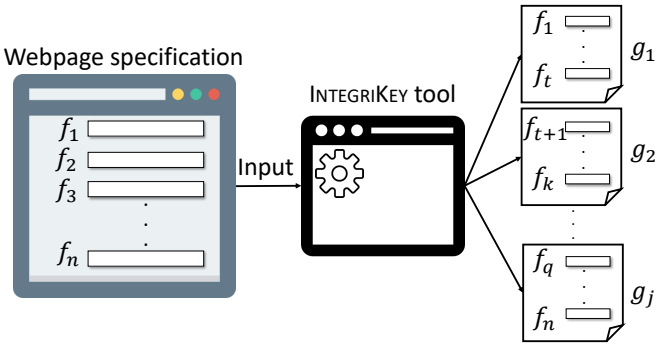

Fig. 4: **High-level flow of operation of the INTEGRIKEY tool at the server side. The tool takes a web page specification with n input fields $f_1, f_2, \ldots f_n$ and converts it to set of subsets of the original fields $g_1, g_2, \ldots, g_j$ such that all the fields in a set (g) are swappable. One concrete example is illustrated in the Figure 3.**

fields whose labels are swappable. We assume that the logic in the server already employs some basic sanity checks to eliminate incorrectly formatted data. This is why we focus on the fields with values that are very closely related, which are very hard for the server to distinguish and thus are susceptible to the input label swapping attack. The developers provide a fine-grained specification to our tool containing information such as data types (`integer`, `string`, `boolean`, etc.), the possible range of values(for number datatype)/lengths (for `string` datatype), the regular expressions, etc.

CLASSIFICATION OF THE INPUT FIELDS. Classification of input fields is critical to understand which input field labels can be swapped by the attacker. Moreover, proper classification of the field will reduce user's load as the user only needs to add the label with the input data corresponding to the fields that are swappable. We design and develop the server-side INTEGRIKEY tool that takes a web-page and developer generated specification to calculate the set of input fields that can be reordered. Figure 4 provides a high level operation of INTEGRIKEY tool. INTEGRIKEY tool takes a web page specification that consists of a set of n input fields $\{f_1, f_2, \ldots, f_n\}$ including their specification. It then outputs a set of fields groups $g_1, g_2, \ldots, g_k$ where each g is a subset of the input fields in the original page that are swappable. E.g., assume input fields f_1, f_2, f_3 where f_1 and f_2 can be swapped, and f_3 is not swappable with any other fields. INTEGRIKEY tool outputs two groups g_1, g_2 where $g_1 = \{f_1, f_2\}$ and $g_2 = \{f_3\}$.

INPUT FIELD SPECIFICATION. Note that the semantic of a field label contribute nothing to the fact whether two different fields are swappable or not, only the characteristics of the input field. We capture the characteristics of an input field with specifications. Trivially, one field is always swappable with the identical one. One example is in the home automation system, the user can set the temperature of a specific room by providing the input to the web application. The attacker can swap the labels of the field with the label of the temperature of another room.

More interesting examples are the fields which are semantically disjoint but share specifications. E.g., the parameters for the medical devices where the doctor can set 'blood pressure' and 'heart rate limit'. As the range of these two fields is overlapping, the attacker can swap the labels of two such fields even though the fields 'blood pressure' and 'heart rate limit' are semantically different. We notice that some fields are strictly swappable only with another identical field. Such as an arbitrary field is not swappable with any other IBAN number due to the specific format (e.g., $(ISO3166-1$ IBAN code$)[0-9A-Z]^+$ with minimum and maximum length of 20 and 30 respectively).

The developer uses the specification to specify the input field data format to the INTEGRIKEY tool. We now illustrate one simple example with numerical fields. Assume that there are input fields f_1, f_2, f_3, all of which are for numerical input fields. Let f^{max} and f^{min} denote the maximum and minimum possible values corresponding to the field f. If one the following conditions hold:

$$f_i^{max} < f_j^{min} \text{ or } f_i^{min} > f_j^{max}$$

Then there exist no values that are the valid inputs to f_i and f_j at the same time. Otherwise, f_i and f_j can be swapped by the attacker. We call this test of finding overlapping values as "swappable test". Algorithm 1 uses such check for numerical fields at line no 12.

The analysis for the `string` type input fields is, however, nontrivial as the INTEGRIKEY tool needs to handle various types of `string` constraints. The tool contains a knowledge database that specifies the regular expression for the well-known fields. One such example is the name field which is an alphabetic type and can be represented as $s[a-zA-Z]^+(\,|[a-zA-Z])^*[min = 1, max > min]$. This implies that the name may contain alphabets from a to z in both upper and lower cases for both the first and last name. If the person also has the last name then it is separated from the first name by an empty space character. Moreover, the specification describes that the minimum (min) length of the `string` can be 1 but there is no restriction on the maximum (max) length. Similarly, the address is an alphanumerical `string` datatype and can be represented as $s[a-zA-Z0-9\ |,|/|-]^+[min = 10, max > min]$, where '|' denotes conjunction.

We now define the formal structure of the specification of an input field as:

$$datype[regx][min = x, max = y]$$

datatype denotes the input datatype such as `string` (s), `integer` (i), `float` (f), `date` (d), `time` (t) and `boolean` (b). $[regx]$ is valid only for the `string` datatype. min and max represents the minimum and maximum length of the data if the datatype is `string`, minimum and maximum value of the data if the datatype is `integer`, `float`, `date` or `time`. An example web page specification is presented in Specification 1 that corresponds to the example illustrated in Figure 3. We use XML to represent the specification. `<Label>` and `<RegEx>` represent the input

Algorithm 1: Server-side mechanism to find set of overlapping input fields

Input: Webpage P with a set of input fields F and the specification S.
Output: Set of subset of fields $G = \{g_1, \ldots, g_n\}$ where all the fields in a $g_i \in G$ are swappable.

```
1  begin
2      G ← Initialize empty group
3      for ∀f ∈ F do
4          for ∀f_in ∈ F do
5              if f ≠ f_in ∧ f.type = f_in, type then
6                  addField ← false
7                  if f.type = string then
8                      f.regEx, f_in.regEx ← read from S
9                      if f.regEx ⊂ f_i n.regEx then
                           addField ← true
10                 end
11                 if f.type = integer ∨f.type = flaot
                     ∨f.type = time ∨f.type = date then
12                     if ¬(f^max < f_in^min ∨ f^min > f_in^max) then
                           addField ← true
13                 end
14                 if f.type = boolean then addField ← true
15                 if addField = true then
16                     g ← empty set of fields
17                     g.add(f, f_in)
18                     G.add(g)
19                     addField ← false
20                 end
21             end
22         end
23     end
24     return G
25 end
```

Specification 1: **Specification of the input fields corresponding to the PLC configuration page illustrated in figure 3**

```
<InputSchema>
 <Input>
   <Label>Relay 1</Label>
   <RegEx>s[a-zA-Z0-9]+[min=1,max>min]</RegEx>
 </Input>
 <Input>
   <Label>Decimal places 1</Label>
   <RegEx>i[0-9]*[min=0,max=5]</RegEx>
 </Input>
 <Input>
   <Label>Temp Relay 1 (deg c)</Label>
   <RegEx>i[0-9]*[min=-20,max=150]</RegEx>
 </Input>
 <Input>
   <Label>Relay 2</Label>
   <RegEx>s[a-zA-Z0-9]+[min=1,max>min]</RegEx>
 </Input>
 <Input>
   <Label>Decimal places 2</Label>
   <RegEx>i[0-9]*[min=0,max=5]</RegEx>
 </Input>
 <Input>
   <Label>Temp Relay 2 (deg c)</Label>
   <RegEx>i[0-9]*[min=-10,max=100]</RegEx>
 </Input>
 <Input>
   <Label>Unit</Label>
   <RegEx>s[unit][min=1, max=5]</RegEx>
 </Input>
</InputSchema>
```

field label and the corresponding regular expression (includes data type and length/value constraints), respectively. The data fields in the web page is the set {`Relay 1`, `Temp relay 1`, `Decimal places 1`, `Relay 2`, `Temp relay 2`, `Decimal places 2`, `Unit`}.

FINDING OVERLAPPING FIELDS. Upon receiving all the specifications, the tool evaluates all the input fields by executing swappable tests on them. The test to find the overlapping values for the `integer` and the `float` type is discussed above. For the `string` datatype, the test involves regular expression swappable criteria. For example, given the following two expressions

$$RE_1 = s[a - zA - Z]^+[min = x, max = y]$$

$$RE_2 = s[a - zA - Z0 - 9]^+[min = x, max = y]$$

$$\implies RE_1 \subsetneq RE_2$$

RE_1 represents a `string` containing uppercase or lowercase alphabetic characters. RE_2 represents a `string` containing uppercase, lowercase alphabetic or numerical characters ranging from 0 to 9. It is clear that RE_1 is a subset of RE_2 as all strings from RE_2 are also members of RE_1 but there are strings in RE_2 (e.g., abc123) that are not in RE_1. This can be verified by checking if $RE_1 \cap (RE_2)^c = \phi \implies RE_1 \subset RE_2$, where ϕ denotes empty set. Algorithm 1 uses this subset check (in line no 9) to determine the overlapping fields. The subset criteria may also change depending on the minimum and maximum length of the specific input fields.

We use the swappable test to design the Algorithm 1 that generates the group containing overlapping input fields. Finding if a regular expression is a subset of another regular expression requires conversion of the regular expression to a deterministic finite automaton (DFA). This has a worst-case exponential [19] ($\mathcal{O}(2^S)$) timing complexity with respect to the number of states (S) in the non-deterministic finite automaton (NFA) that is derived from the regular expression. Moreover, the algorithm requires computing pairwise swappable tests over all the input fields in a page. This is quadratic $\mathcal{O}(|F|^2)$ with respect to the number of fields ($|F|$). Therefore, the timing complexity of Algorithm 1 is $\mathcal{O}(|F|^2 2^{|S|})$.

INTEGRIKEY tool takes the specification provided in Specification 1 and produces two groups: g_1 ={`Relay 1`, `Relay 2`} and g_2 ={`Temp Relay 1`, `Temp Relay 2`, `Decimal places 1`, `Decimal places 2`} that are overlapping set of fields while the `Unit` field is distinct from the rest. Note that the precise calculation of the overlapping groups of fields requires the developer to provide a tight/well-defined specification otherwise, the INTEGRIKEY tool may over-approximate.

Upon finding the set of overlapping fields, the server embeds an instruction in the web page, instructing the user to add the label name with the input data. Note that this requires the server to change the type of the field to accommodate the label name (such as a `date` type field to a `string` type

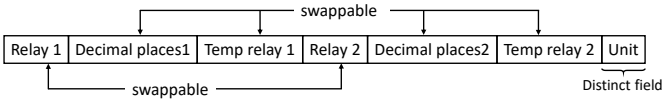

Fig. 5: **An example what the server expects from the browser when the user submits a form. We have used the specification illustrated in the Specification 1. This figure indicates the group of fields that are swappable.**

field). E.g., for a numerical data for a field `Relay temp 1` the user provides $Relay\ temp1 : 20$.

SERVER-SIDE VERIFICATION. Server-side verification executes after the user submits the form on the browser. For a specific web page with a set of n input fields $\{f_1, f_2, \ldots, f_n\}$, the server expects to receive n inputs via the HTTPS payload when the user submits the form. In this form, only the overlapping fields require the label of the field to be added to the data. The rest do not require the label as they have the distinct specification. For our example scenario (the web page is illustrated in Figure 3 and the specification is provided in the Specification 1), the server expects that the user provides the label information for the fields `Relay 1` and `Relay 2` as these are swappable and also for the group of fields `Decimal places 1`, `Decimal places 2`, `temp relay 1` and `temp relay 2`. As the field `Unit` is a distinct field, the server does not expect a label added with the data as the field is not swappable with any other field. Additionally, the server also receives the signed data from the INTEGRIKEY bridge (via the dedicated TLS channel) for verification. Upon receiving the data the server executes the following steps

(**1**) First the server checks the number of field data corresponding to the webpage. If it matches with the number of input from the INTEGRIKEY bridge, the server proceeds to the next step. There may exist some optional fields that are not required to be filled by the user.

(**2**) After that, the server matches the data from the browser (HTTPS) and the INTEGRIKEY bridge (TLS) and executes the signature verification. If they match, the server proceeds to the next step. Otherwise, it rejects the data and sends a feedback to the INTEGRIKEY bridge.

(**3**) Upon successful matching of the data received from the browser and the INTEGRIKEY bridge, the server searches for the input field labels that are added to the data. When found, the server splits the data from the label and parses it. After parsing, the server checks if the data satisfies the specification that is provided to the INTEGRIKEY tool (by matching the data with the regular expression and value/length constraint in the specification). In case the server does not find the label with the data for the overlapping fields, it rejects the data.

B. User Side Mechanisms

The INTEGRIKEY bridge, which is connected to the host system at the user's side, executes the following steps:

(**1**) As described in the previous section, the browser creates a HTTPS (TLS_1) connection to the remote server. INTEGRIKEY bridge creates another TLS channel (TLS_2) with the remote server by leveraging WebUSB API.

(**2**) The server sends the web page containing the input fields to the browser. The server also embeds a text message specifying the input fields that require the user to add the field label. The user fills up the forms.

(**3**) When the user submits the form, the browser sends the form data through the HTTPS (TLS_1) connection and the INTEGRIKEY bridge sends the data via the TLS_2.

C. Security Analysis

ATTACKER MODEL. As before, we assume a strong attacker that compromises the host system and the network. As the host system is compromised, the attacker can intercept and manipulate any user input and change any visual elements displayed to the user. In Section III, we address the basic attack where the attacker only changes the input data provided by the user through the input peripherals. In this section, we concentrate on the attack where the attacker swaps the labels of the input fields arbitrarily in a single web page or across a set of web pages.

ANALYSIS. We introduce the INTEGRIKEY tool in Section IV-A to analyze the web pages and calculate the overlapping input fields from the developer given specification. After the analysis, the server includes a text message on the web page instructing the user to add the label to the input filed with the input data.

The fields that are not overlapping with any other input field do not require the user to add the label. If the attacker executes the swapping attack on such fields, the server can easily recognize such data (due to their distinct specification) and reject them. For the rest of the overlapping fields, the user provides a (label, data) pairing of the input data. Note that the INTEGRIKEY bridge sends the signed (label, data) pair using the dedicated TLS channel. This ensures that any manipulation attempt by the malicious host would readily be detected by the server. Swapping across the set of web pages can be prevented if the developer provides the combined specifications sampled from a number of web pages together.

V. IMPLEMENTATION

We implement a fully functional INTEGRIKEY prototype that consists of i) the INTEGRIKEY bridge based on an Arduino board, ii) the JavaScript code snippet that uses Google Chrome's WebUSB API to handle the communication between the remote server and the INTEGRIKEY bridge, and iii) the remote server, which provides a webpage with input forms. Figure 6 shows the INTEGRIKEY bridge prototype.

EXPERIMENTAL SETUP. All the experiments were performed on a laptop with a 3.2 GHz quad-core Intel i5 CPU and 16 GB of main memory running Ubuntu 16.10 64-bit. We use Google Chrome version 61 and JDK v1.8 with 4GB heap space for all the experiments.

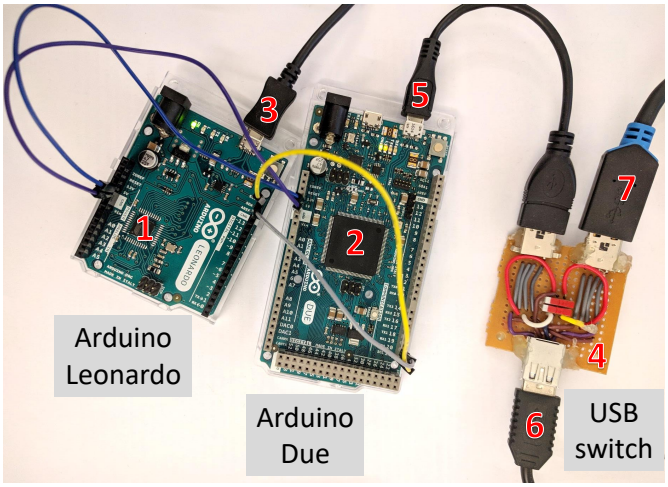

Fig. 6: **Prototype implementation of INTEGRIKEY bridge, consisting of the following: 1) an Arduino Due board that is connected with the USB peripheral and executes cryptographic operations in TLS, 2) an Arduino Leonardo board that communicates with the browser using WebUSB, 3) a USB connection from the INTEGRIKEY bridge to the host system, 4) a USB switch to switch between the secure and insecure mode (the insecure mode is a pass-through), 5) the connection between the INTEGRIKEY bridge and the USB switch, 6) the keyboard connection, 7) the host pass-through connection for the insecure mode.**

A. INTEGRIKEY *bridge*

Figure 6 shows the INTEGRIKEY bridge prototype setup and all the necessary modules. We build our hardware prototype on top of the Arduino prototyping boards. The following are the connections that connect INTEGRIKEY bridge to the host and a USB input device.

(**1**) The INTEGRIKEY bridge is connected to the host system via USB interface (3 in the Figure 6).

(**2**) The INTEGRIKEY bridge is connected to the USB input devices such as keyboard via the USB interface (6 in the Figure 6).

Our prototype implementation of the INTEGRIKEY bridge uses an Arduino Leonardo board (1 in Figure 6, 16 MHz AVR microcontroller) to communicate with the host system using WebUSB and an Arduino Due (2 in Figure 6, 84 MHz ARM Cortex-M3 microcontroller) to execute cryptographic operations required for TLS. We used these board in master (Leonardo)-slave (Due) configuration over the I^2C [20] protocol. The INTEGRIKEY bridge takes input from the keyboard using the `USBHost` library. As the current version of the WebUSB library allows only one USB interface, the INTEGRIKEY bridge cannot emulate a keyboard (interrupt transfer) and a persistent data (bulk transfer) device required for the TLS channel at the same time. Therefore, the INTEGRIKEY bridge sends keyboard signals to the JavaScript code running in the browser. The JavaScript code interprets these signals and translates them to keyboard input on the web page. This gives an impression to the user that the device emulates a keyboard. The keyboard is connected with the Due board via a custom built hardware switch (4 in Figure 6). Users can toggle between the *secure* mode (input processed by the device) and the *insecure* mode (pass-through to the host system). We use the Arduino cryptographic library for the TLS. The TLS includes 128 bit AES as the symmetric cipher in counter mode (CTR), Ed25519 & Curve25519 for elliptic curve digital signature & Diffie-Hellman key exchange respectively and SHA256 for the cryptographic hash function. The limited set of cipher suites helps make the code fit into the limited program space. The INTEGRIKEY prototype code-base is ~ 3 KLOC (kilo lines of code).

B. *Remote Server*

The remote server is written using the JAVA EE Servlet API and hosted on an Apache Tomcat web server. The server also implements a similar reduced TLS stack like the INTEGRIKEY bridge, using the JAVA cryptographic API. The code size of the INTEGRIKEY server-side component is around 500 lines, making it easy to deploy. For specific test cases such as the web-based PLC, we use ControByWeb $x600m$ industrial I/O web server. Figure 7 shows one such configuration page from the server.

C. *WebUSB JavaScript code*

The remote server sends a JavaScript code when the user loads a specific website in her browser. The JavaScript code communicates with the INTEGRIKEY bridge and establishes a TLS channel. We develop this JavaScript code that uses Google Chrome's WebUSB API to enumerate and communicate with the INTEGRIKEY bridge. We use `XMLHttpRequest` to communicate with the remote server. As the attacker has full control over the browser, he can also manipulate the JavaScript code. Therefore, no cryptographic operations are performed inside the JavaScript code.

D. INTEGRIKEY *Tool*

The INTEGRIKEY server-side tool is written in JAVA based on the JAVA AWT graphics library. The tool is around 1.5 KLOC and uses the JAVA native XML interpreter library to read the specification and JSOUP HTML parser to parse web pages.

VI. EVALUATION

We now evaluate the performance of individual modules of INTEGRIKEY to demonstrate its feasibility.

A. INTEGRIKEY *tool*

We consider a number of browser-based web applications and analyze their web pages for overlapping input fields. These applications include PLC controllers, home automation systems, medical device control, personal data management and online banking. In Table I, we show the input fields of the aforementioned applications, the data types of the input fields, the regular expressions of the input format, and the constraints on length (for `string`), values (for `integer`, `float`, and `boolean` data types) or ranges (for `date` and

TABLE I: **Specification (columns 2, 3, 4) of different input fields sampled from different web-applications. The reorder column denotes that the group of input fields with ✓ mark can be swapped with each other. Such as the current can be swapped with the frequency field. It is trivial that all the input fields can be swapped with another identical field.**

Name	Type	Regular expression	Length/value constraints	Reorder
Personal information				
Email	string	$(*)^+(@)[a-zA-Z0-9]^+(.)[a-z]^+$	$[min=5, max=*]$	
Name	string	$[a-zA-Z.]^+$	$[min=1, max=*]$	
Address	string	$[a-zA-Z0-9]^+$	$[min=5, max=*]$	
Financial transaction				
IBAN account no.	string	$(ISO3166-1$ IBAN code$)[0-9A-Z]^+$	$[min=20, max=30]$	
Transaction amount.	float	$(ISO4217$ currency code$)[0-9]^+((.)[0-9])^*$	$[min=0, max=*]$	
Medical parameters				
Heartbeat	integer	$[0-9]^+$	$[min=55, max=210]$	✓
Body temperature	float	$[0-9]^+((.)[0-9])^*$	$[min=94, max=108]$	
Blood pressure	integer	$[0-9]^+$	$[min=80, max=150]$	
Blood sugar (Fasting)	integer	$[0-9]^+$	$[min<108, max>126]$	
Web-based PLC form				
Thermocouple	integer	$[0-9]^+$	$[min=-15, max=150]$	✓
Frequency	integer	$[0-9]^+$	$[min=0, max=500(Hz)]$	
Analog Input(voltage)	float	$[0-9]^+((.)[0-9])^*$	$[min=0, max=12]$	
Current	float	$[0-9]^+((.)[0-9])^*$	$[min=300(mA), max=2(A)]$	
Logic repetition	integer	$[0-9]^+$	$[min=0, max=9999]$	
Event duration	integer	$[0-9]^+$	$[min=0, max=9999999999]$	
Decimal places	integer	$[0-9]^+$	$[min=0, max=5]$	
Initial value	integer	$[0-9]^+$	$[min=0, max=999999]$	
Relay status Thermocouple status Thermocouple status Energy slave status Input module status Thermostat status	boolean	$(0\|1)$	$[min=0, max=1]$	✓
Logic start/end date	date	$[0-9]^+(/)[0-9]^+(/)[0-9]^+$	$[min=1/1/2007, max=12/12/2029]$	
Logic start/end time	time	$[0-9]^+(:)[0-9]^+$	$[min=00:00:00, max=23:59:59]$	
Logic Script	string	$(*)^+$	valid controller script	
Module name	string	$[a-zA-Z0-9]^+$	$[min=1, max=20]$	✓
Description			$[min=0, max=60]$	
Web-based home automation				
Room light toggle Door lock toggle Alarm A/C	boolean	$(0\|1)$	$[min=0, max=1]$	✓
Room temperature	integer	$[0-9]^+$	$[min=6, max=25]$	
Window shutter level	integer	$[0-9]^+$	$[min=0, max=8]$	
Alarm time	time	$[0-9]^+(:)[0-9]^+$	$[min=00:00, max=23:59]$	

TABLE II: **INTEGRIKEY tool performance.**

Web page	#Fields	Processing time (ms.)	SD
x600m Web PLC			
Register configuration	6	1.654	0.0131
Counter configuration	7	0.771	0.0089
Event configuration	8	0.622	0.0085
Action configuration	5	1.241	0.0111
Supply voltage	4	0.673	0.0099
Calender configuration	11	0.713	0.0105
Home automation			
Home configuration	6	0.016	0.0018
Room configuration	5	0.012	0.0015

TABLE III: **INTEGRIKEY bridge performance.**

Parameter	Latency (ms.)	SD
Initial secure channel establishment	800	0.1235
Keystroke latency	50	0.0535

TABLE IV: **INTEGRIKEY server-side latencies.**

Parameter	Latency (μs)	SD
SHA256 Hash matching (512 bytes message)	5.359	0.0303
SHA256 Hash matching (1 Kbytes message)	9.203	0.0479
Curve25519 signature verification	197.856	0.6487

time data types). The "Reorder" column specifies if a group of input fields additionally can be swappable with each other.

From the initial evaluation, we found that all boolean data fields are vulnerable to the label swapping attack. In most of the pages, the boolean type inputs are interpreted by either a toggle button or a check-box. Converting those UI elements to text-fields that accepts either 0 or 1, with the label of the field added to the data can prevent such attacks.

Whether the string type fields are swappable depends on both their lengths and their regular expressions. E.g., a PLC application that supports Lua scripts, allowing the administrator to write a script that executes operations based on the sensor data (e.g., heat, humidity sensor). Parameters such as the device name and descriptions are also string type but cannot be swapped in place of the script since they typically do not follow the syntax of Lua. Numerical data types such as integer and float are susceptible to swapping attacks due to their overlapping values. E.g., the register configuration on the PLC server, the initial value of the register and decimal places are both of the integer types (see Table V) and

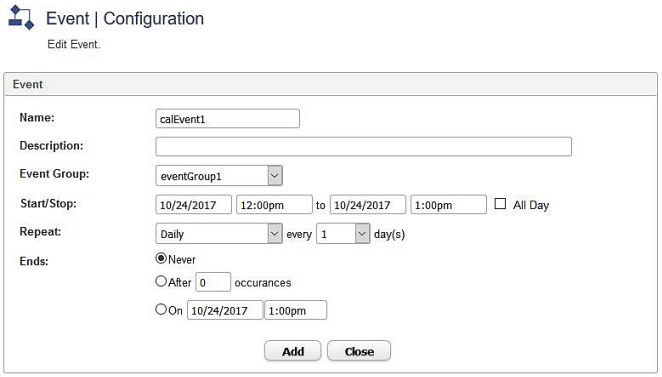

Fig. 7: **Example screenshot of a ControlByWeb x600m industrial I/O server configuration webpage in the browser.**

there is an overlap in their value ranges, $\{0 - 99999\}$ and $\{0 - 5\}$, respectively. Even though the overlapping range only $\{0 - 5\}$, it may lead to catastrophic results if the attacker successfully tricks the user into entering them in the wrong fields. There are some fields such as, start date end dates which are classified by the INTEGRIKEY tool as overlapping fields. However, interchanging them is not possible by the attacker as the server will reject such invalid values. Note that this is due to the fact the INTEGRIKEY tool is oblivious to any of the internal logic of the application, rather solely depends on the developer's specifications. Hence, such over-approximation could be avoided by providing a tighter bounds/exceptions.

We evaluate the INTEGRIKEY tool with two web applications: the PLC and the home automation controller. We first generate the specifications based on the original web pages of these applications. Then we feed the specifications to the INTEGRIKEY tool to produce the groups of overlapping fields. Table V provides the details of the evaluation. For the PLC test case, we use ControlByWeb *x600m* [2] I/O server and select six configuration pages from it. We manually generate the specification for the data fields in these pages as an input to the INTEGRIKEY tool. Similarly, we select two configuration pages from a browser-based home automation system provided by home-assistant [21]. Then we apply the INTEGRIKEY tool to compute the set of overlapping fields.

B. INTEGRIKEY *bridge performance*

We evaluate the performance of the INTEGRIKEY bridge in the following two aspects:

(**1**) **Secure channel establishment:** This represents the elapsed time between the web page loads and the INTEGRIKEY bridge is ready to take input from the user. The JavaScript code served by the remote server communicates with the INTEGRIKEY bridge and establishes a TLS using the WebUSB API. The additional TLS messages and the INTEGRIKEY bridge processing introduce this delay only at the initial loading of the page. We measure the difference between the time when the JavaScript code gets loaded on the browser and the time when the final TLS handshake message from the client is sent using `performance.now()` that provides the highest resolution for timing measurement.

(**2**) **Keystroke latency:** This latency is added every time the user presses a key. This time is due to the internal processing of the INTEGRIKEY bridge. We use the `micros()` function that is provided by the `Arduino` library which measures elapsed times in microsecond resolution. We place the `micros()` at program point the `USBHost` library starts capturing the keyboard event and at the program point the device calls `WebUSBSerial.print()`, a function that sends the data via the WebUSB interface to the browser.

Table III provides the two aforementioned latencies. We use *http-archive* [22] and *pingdom* [23] to evaluate the loading time of some of the Alexa top 100 [24] websites. We found that `google.com` takes 1.1s, `Wikipedia.org` takes 1s, `youtube.com` takes 2s, `baidu.com` 5.47s, `facebook.com` takes 1.08s, and `cnn.com` takes 30s. INTEGRIKEY bridge adds small overhead compared to the actual loading time of the popular pages. The latency corresponding to the keystroke is around 50 millisecond which is negligible. These latencies are very specific to the INTEGRIKEY bridge implementation. The I^2C channel between the master and the slave device is only limited to 1 kHz. We have also started development on a standalone Arduino/Genuino Zero that is now supported by the new version of the WebUSB driver. This new implementation eliminates the need for the I^2C channel and can potentially reduce the latencies significantly.

C. INTEGRIKEY *server side performance*

INTEGRIKEY TOOL. For the performance measurement of the INTEGRIKEY tool, we use identical test cases provided in the Table V. Table II represents the performance statistics for detection of the overlapping fields. The time required to execute the performance depends on i) the number of states in the DFA constructed from the regular expression of the specification and ii) the number of input fields. We conclude that the time required to do the processing is extremely short (the highest one is 1.656 ms). Moreover, this processing is static and can be done by the developer at the design phase and does not incur an overhead during runtime.

INTEGRIKEY SERVER-SIDE COMPONENT. We use Apache Tomcat web server for all our test cases. We measure the time taken by the server to i) compare the values coming from the browser (HTTPS) and the dedicated channel (TLS) between the server and the INTEGRIKEY bridge, ii) verify the signatures (Curve25519) of the messages. Table IV provides the performance numbers for the server-side components. The signature verification takes ~ 198 μs and the hash matching (for 512 bytes messages) takes ~ 5 μs.

VII. DISCUSSION

We now discuss various aspects of INTEGRIKEY.

INTEGRIKEY DEPLOYMENT. One critical aspect of the design of INTEGRIKEY is its compatibility and seamless integration with the legacy systems while incurring minimal or no cognitive load on the users. Our prototype implementation

TABLE V: **Evaluation of INTEGRIKEY tool on 'ControlByWeb x600m' industrial I/O server and 'home-assistant' home automation systems. Page column shows the actual pages that we sampled from the applications and generates their specifications. Field and type are the corresponding input fields and datatype. Overlapping fields shows which field label can be swapped with another. `menu` types are drop-down menus that are not swappable with any other data-types.**

Web pages	Fields	Type	Length/value constraint	Overlapping fields
Web PLC configuration forms				
Register configuration	Name	string	$[min = 1, max = 20]$	Name
	Description	string	$[min = 0, max = 60]$	Description
	Type	integer	$[min = 1, max = 5]$	Units
	Units	string	$[min = 1, max = 5]$	Decimal place
	Decimal places	integer	$[min = 0, max = 5]$	Initial
	Initial Value	integer	$[min = 0, max = 999999]$	Type
Counter configuration	Device	boolean	$[min = 0, max = 1]$	Name
	Device counter number	integer	$[min = 0, max = 50]$	Description
	Name	string	$[min = 1, max = 20]$	Device counter number
	Description	string	$[min = 0, max = 60]$	Decimal places
	Decimal places	integer	$[min = 0, max = 5]$	Debounce
	Debounce	integer	$[min = 0, max = 9999]$	Edge
	Edge	integer	$[min = 0, max = 6]$	
Event configuration	Name	string	$[min = 1, max = 20]$	Name
	Description	string	$[min = 0, max = 60]$	Description
	Type	menu	–	
	I/O	menu		
	Event group	menu		
	Condition	menu		
	Eval on powerup	boolean	$[min = 0, max = 1]$	
	Duration	integer	$[min = 0, max = 9999]$	
Action configuration	Name	string	$[min = 1, max = 20]$	Name
	Description	string	$[min = 0, max = 60]$	Description
	Event source	menu	–	
	Type	string	$[min = 0, max = 5]$	
	Relay	menu	–	
Supply voltage	Name	string	$[min = 1, max = 20]$	Name
	Description	string	$[min = 0, max = 60]$	Description
	Decimal places	integer	$[min = 0, max = 5]$	
	Device	menu	–	
Calendar configuration	Name	string	$[min = 1, max = 20]$	Name
	Description	string	$[min = 0, max = 60]$	Description
	Event group	integer	$[min = 0, max = 5]$	Start date
	Start date	date	$[min = 01/01/2007, max = 31/12/2029]$	Stop date
	Stop date	date	$[min = 01/01/2007, max = 31/12/2029]$	Start time
	Start time	time	$[min = 00:00, max = 23:59]$	Stop time
	Stop time	time	$[min = 00:00, max = 23:59]$	Occurrence
	All day	boolean	$[min = 0, max = 1]$	Repeat val
	Repeat type	menu	–	
	Repeat val	integer	$[min = 10, max = 9999]$	
	Occurrences	integer	$[min = 0, max = 999999]$	
Web Home automation configuration forms				
Home configuration	Room door lock	boolean	$[min = 0, max = 1]$	Room door lock
	Alarm	boolean		Alarm
	Water lawn	boolean		Water lawn
	Alarm time	time	$[min = 00:00, max = 23:59]$	
	Nest (thermostat)	integer	$[min = 16, max = 25]$	
	Sound selection	menu	–	
Room configuration	Table lamp	boolean	$[min = 0, max = 1]$	Table lamp
	TV back light	boolean		TV back light
	Celling lights	boolean		Celling lights
	AC	integer	$[min = 16, max = 25]$	
	Window shutter level	integer	$[min = 0, max = 10]$	

shows that INTEGRIKEY can be easily deployed on a large scale for the following reasons:

- As we envision that WebUSB will be supported out-of-the-box by all browsers, the setup of the INTEGRIKEY bridge would require no additional software on the host system.
- The INTEGRIKEY server-side component introduces minimal changes on the server. Only the JavaScript code that handles the WebUSB functionality needs to be included in the web page of the web server. This requires one additional line in the web page i.e., `<script>jsFile.js</script>`.
- The INTEGRIKEY tool is a preprocessing tool that the developer uses at the time of designing the web page. Thus, it does not add any overhead to the online operation of the server.

OTHER USE-CASES. INTEGRIKEY has the potential in other use cases such as the authenticated logging of user input in the context of intrusion detection. The INTEGRIKEY bridge can act as a keylogger that stores the commands given by the users. This data can be stored in the secure internal storage of the INTEGRIKEY bridge and can be presented as a proof later on. More importantly at the time of providing the commands to the remote server, the INTEGRIKEY bridge

can authenticate the user as the valid administrator besides conventional credentials like passwords. Apart from the web-based interfaces, INTEGRIKEY can be used in any applications that are used to configure remote systems. One only has to provide the mechanism to communicate with the (USB based) INTEGRIKEY bridge, similar to WebUSB in browsers) for secure communication with the remote server.

USAGE OF WEBBLUETOOTH. WebBluetooth is also an upcoming web standard like WebUSB that allows a JavaScript code to communicate with Bluetooth enabled devices. INTEGRIKEY can also be implemented using WebBluetooth by attaching a Bluetooth module to the INTEGRIKEY bridge. This setup allows the INTEGRIKEY bridge to communicate with the browser over Bluetooth. Alternatively, smartphones can be used as the INTEGRIKEY bridge (discussed in Section III-D) that connects to the host system over the Bluetooth channel.

VIII. RELATED WORK

We now summarize related works on improving the integrity of control data and compare them with our approach.

PLC SECURITY. Garcia et al. [25] explore the security of programmable logic controllers that are typically deployed in physical infrastructures such as the power grid. Their attack, HARVEY, consists of a PLC rootkit that modifies legitimate outward control commands and inward sensor data to covertly damage power equipment. As a mitigation, they propose using remote software attestation, secure booting, and a bump-in-the-wire device to detect malicious modifications. In contrast, our approach secures control commands sent from users (e.g. administrators in the control center) and therefore require no detection of malicious input on the channel.

Zonouz et al. [26] propose a way detect malware in industrial control devices by automatically analyzing PLC software. They designed the Trusted Safety Verifier (TSV) that runs on a minimal trusted computing base and verifies the PLC code against predefined safety checks before executing it. Malchow et al. [27] present PLC Guard as an improvement to TSV that allows engineers to inspect changes between different versions of PLC code to identify potential threats. Fachkha et al. [28] provide a complete summary of security solutions for cyber-physical systems, showing that most of the work on improving PLC security by analyzing/detecting protocol vulnerabilities [29]–[34], verifying safety-critical codes [35], process variables [36]–[39], and sensor data [40], [41]. By comparison, our work focuses on administrators' control input and defends against a compromised host that attempts to send malicious administrative commands to benign PLCs.

USB SECURITY. As USB is a ubiquitous interface, many security concerns arrive. The paper [42] talks about a system USBSec that incorporates host authentication to defend against software threats. The paper [43] talks about the end-point security of USB device in case of theft. Bang et al. [44] provided the design and implementation of a secure USB bypassing tool that bypasses the USB security functions. Tian et al. propose GoodUSB [45] and USBFilter [46] that provide protection against malicious USB firmware.

COMPROMISED HOSTS. Lin et al. in their paper [12] describe one of the first known fabrication time attacks. Yang et al. in their paper A2 [13] describes a fabrication time attack that leverages analog circuits to create a small and stealthy hardware attack. The paper, ghost on the browser [47] analyzes browser-based malware. Man in the browser [48] describes attacks such as stealing authentication and altering transaction data to benefit the attacker. Drive-by-download is another well-known way to infect victim host systems with malware. Papers such as [49]–[56] analyze and propose mitigation techniques. Zero-day exploits are one of the most effective ways to compromise hosts where the exploits are completely unknown to the hardware/software manufacturers. Bilge et al. in their paper [14] conducted an empirical study on the zero-day attacks in the real world. Other papers such as [6], [15] discussed malware based on the zero-day exploits and the markets of the zero-day exploits. Grace et al. [57] proposed defenses against android based zero-day exploits. Other solutions/detection techniques can be found in like: [58]–[61].

SECOND-FACTOR SECURITY. Many other works consider the use of a device as an additional factor to improve security guarantees. Mannan et al. [62] proposed to use a personal device (e.g., phone) to provide long-term low entropy secret such as the password. They also proposed a protocol MP-auth and implemented a prototype. Kiljan et al. [63] propose a device where the user provides a single transaction information, it signs the values and give it to the computer. The solution requires the user to look at the display of the external device and confirm the values. The proposed method is vulnerable to replay attack and does not consider any UI manipulation or compromised host. Several other second factor authentication proposals can be found in [64]–[69].

IX. CONCLUSION AND FUTURE WORK

We designed, implemented, and evaluated INTEGRIKEY, a system that provides end-to-end integrity for peripheral devices. We identified and analyzed a new form of attack—the UI integrity manipulation attack—that can be orchestrated by a malicious host by manipulating the web page elements shown in the browser. INTEGRIKEY preserves the integrity of user input in the presence of such attacks. INTEGRIKEY is very easy to deploy, making it very practical to use in legacy and modern systems alike, such as industrial PLCs, medical programmers, home automation systems, etc. The INTEGRIKEY bridge can be used in many client systems and be paired with any USB based peripheral devices. Our evaluation showed that the INTEGRIKEY server-side component and the INTEGRIKEY bridge on the user's side introduce only minimal overhead. Currently, INTEGRIKEY is limited to character-based input devices such as the keyboard. As future work, we aim to explore the integration of other input devices, such as mice, touchscreen and more complex UI elements.

REFERENCES

[1] B. Mahato, T. Maity, and J. Antony, "Embedded web plc: A new advances in industrial control and automation," in *2015 Second Inter-*

national Conference on Advances in Computing and Communication Engineering.

[2] "X-600m — web enabled i/o controller." [Online]. Available: https://www.controlbyweb.com/x600m

[3] "Sitraffic smartguard." [Online]. Available: https://www.siemens.com/global/en/home/products/mobility/road-solutions/traffic-management/strategic-management-and-coordination/centrals/smartguard.html

[4] "Simatic s7-1200." [Online]. Available: https://www.siemens.com/global/en/home/products/automation/systems/industrial/plc/s7-1200.html

[5] "Modicon momentum." [Online]. Available: https://www.schneider-electric.us/en/product-range/535-modicon-momentum/

[6] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005.

[7] S. Granger, "Social engineering fundamentals, part i: hacker tactics," *Security Focus, December*, 2001.

[8] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, 2011.

[9] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[10] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson, "Clickjacking: Attacks and defenses." in *USENIX security symposium*, 2012.

[11] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting frame busting: a study of clickjacking vulnerabilities at popular sites," *IEEE Oakland Web*, 2010.

[12] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson, *Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering*.

[13] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *2016 IEEE Symposium on Security and Privacy (SP)*.

[14] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[15] S. Egelman, C. Herley, and P. C. Van Oorschot, "Markets for zero-day exploits: Ethics and implications," in *Proceedings of the 2013 workshop on New security paradigms workshop*. ACM.

[16] [Online]. Available: https://wicg.github.io/webusb/

[17] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo, "Rethinking connection security indicators," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. Denver, CO: USENIX Association, 2016, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/porter-felt

[18] A. P. Felt, R. W. Reeder, H. Almuhimedi, and S. Consolvo, "Experimenting at scale with google chrome's ssl warning," in *ACM CHI Conference on Human Factors in Computing Systems*, 2014.

[19] K. Salomaa and S. Yu, *NFA to DFA transformation for finite languages*. Springer Berlin Heidelberg, 1997.

[20] "I2c." [Online]. Available: https://learn.sparkfun.com/tutorials/i2c

[21] "Home assistant demo." [Online]. Available: https://home-assistant.io/demo/

[22] "Http archive - web sites." [Online]. Available: http://httparchive.org/websites.php

[23] "Pingdom tools." [Online]. Available: https://tools.pingdom.com/

[24] "The top 500 sites on the web." [Online]. Available: https://www.alexa.com/topsites

[25] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! attacking plcs with physical model aware rootkit," in *24th Annual Network & Distributed System Security Symposium (NDSS)*, 2017.

[26] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated plc code analytics," *IEEE Security & Privacy*.

[27] J.-O. Malchow, D. Marzin, J. Klick, R. Kovacs, and V. Roth, "Plc guard: A practical defense against attacks on cyber-physical systems," in *Communications and Network Security (CNS), 2015*.

[28] C. Fachkha, E. Bou-Harb, A. Keliris, N. Memon, and M. Ahamad, "Internet-scale probing of cps: Inference, characterization and orchestration analysis," in *Proceedings of NDSS*, 2017.

[29] I. Garitano, R. Uribeetxeberria, and U. Zurutuza, *A Review of SCADA Anomaly Detection Systems*.

[30] C. Bellettini and J. L. Rrushi, "Vulnerability analysis of scada protocol binaries through detection of memory access taintedness," in *2007 IEEE SMC Information Assurance and Security Workshop*.

[31] E. Byres, D. Huffman, and N. Kube, "On shaky ground-a study of security vulnerabilities in control protocols," American Nuclear Society, 555 North Kensington Avenue, La Grange Park, IL 60526 (United States), Tech. Rep., 2006.

[32] A. Treytl, T. Sauter, and C. Schwaiger, "Security measures for industrial fieldbus systems-state of the art and solutions for ip-based approaches," in *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*.

[33] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for scada networks," in *Proceedings of the SCADA security scientific symposium*, 2007.

[34] N. Goldenberg and A. Wool, "Accurate modeling of modbus/tcp for intrusion detection in scada systems," *International Journal of Critical Infrastructure Protection*, 2013.

[35] S. E. McLaughlin, S. A. Zonouz, D. J. Pohly, and P. D. McDaniel, "A trusted safety verifier for process controller code." in *NDSS*, 2014.

[36] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel, "Through the eye of the plc: semantic security monitoring for industrial processes," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014.

[37] S. McLaughlin and P. McDaniel, "Sabot: specification-based payload generation for programmable logic controllers," in *Proceedings of the 2012 ACM CCS*.

[38] M. Caselli, E. Zambon, J. Petit, and F. Kargl, "Modeling message sequences for intrusion detection in industrial control systems," in *International Conference on Critical Infrastructure Protection*. Springer, 2015.

[39] S. E. McLaughlin, "On dynamic malware payloads aimed at programmable logic controllers." in *HotSec*, 2011.

[40] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09.

[41] F. Pasqualetti, F. Drfler, and F. Bullo, "Attack detection and identification in cyber-physical systems," *IEEE Transactions on Automatic Control*, 2013.

[42] Z. Wang and A. Stavrou, "Attestation & authentication for usb communications," in *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*.

[43] M. Fabian, "Endpoint security: Managing usb-based removable devices with the advent of portable applications," in *Proceedings of the 4th Annual Conference on Information Security Curriculum Development*, ser. InfoSecCD '07. ACM.

[44] J. Bang, B. Yoo, and S. Lee, "Secure usb bypassing tool," *digital investigation*, 2010.

[45] D. J. Tian, A. Bates, and K. Butler, "Defending against malicious usb firmware with goodusb," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015. ACM.

[46] D. J. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor, "Making USB great again with USBFILTER," in *25th USENIX Security Symposium (USENIX Security 16)*.

[47] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu *et al.*, "The ghost in the browser: Analysis of web-based malware." *HotBots*, 2007.

[48] T. Dougan and K. Curran, "Man in the browser attacks," *International Journal of Ambient Computing and Intelligence (IJACI)*, 2012.

[49] N. P. P. Mavrommatis and M. A. R. F. Monrose, "All your iframes point to us," 2008.

[50] K. Rieck, T. Krueger, and A. Dewald, "Cujo: efficient detection and prevention of drive-by-download attacks," in *Proceedings of ACM CCS 2010*.

[51] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010.

[52] M. Egele, E. Kirda, and C. Kruegel, "Mitigating drive-by download attacks: Challenges and open problems." in *iNetSeC*. Springer, 2009.

[53] F.-H. Hsu, C.-K. Tso, Y.-C. Yeh, W.-J. Wang, and L.-H. Chen, "Browserguard: A behavior-based solution to drive-by-download attacks," *IEEE Journal on Selected areas in communications*, 2011.

[54] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda, "Defending browsers against drive-by downloads: Mitigating heap-spraying code injection

attacks," *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2009.

[55] V. L. Le, I. Welch, X. Gao, and P. Komisarczuk, "Anatomy of drive-by download attack," in *Proceedings of the Eleventh Australasian Information Security Conference-Volume 138*. Australian Computer Society, Inc., 2013.

[56] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011.

[57] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012.

[58] G. Portokalidis and H. Bos, "Eudaemon: Involuntary and on-demand emulation against zero-day exploits," in *ACM SIGOPS Operating Systems Review*. ACM, 2008.

[59] L. Wang, S. Jajodia, A. Singhal, and S. Noel, "k-zero day safety: Measuring the security risk of networks against unknown attacks." in *ESORICS*. Springer, 2010.

[60] C. Leita, M. Dacier, and G. Wicherski, "Sgnet: a distributed infrastructure to handle zero-day exploits," *Institut Eurecom, France, Tech. Rep. EURECOM*, 2007.

[61] G. Portokalidis, A. Slowinska, and H. Bos, "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation," in *ACM SIGOPS Operating Systems Review*, 2006.

[62] M. Mannan and P. C. Van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *International Conference on Financial Cryptography and Data Security*. Springer, 2007.

[63] S. Kiljan, H. Vranken, and M. V. Eekelen, "What you enter is what you sign: Input integrity in an online banking environment," in *2014 Workshop on Socio-Technical Aspects in Security and Trust*.

[64] A. T. B. Jin, D. N. C. Ling, and A. Goh, "Biohashing: two factor authentication featuring fingerprint data and tokenised random number," *Pattern recognition*, 2004.

[65] N. Gunson, D. Marshall, H. Morton, and M. Jack, "User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking," *Computers & Security*, 2011.

[66] N. L. Clarke and S. Furnell, "Advanced user authentication for mobile devices," *computers & security*, 2007.

[67] B. Adida, "Beamauth: two-factor web authentication with a bookmark," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007.

[68] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012.

[69] M. K. Khan and K. Alghathbar, "Cryptanalysis and security improvements of two-factor user authentication in wireless sensor networks," *Sensors*, 2010.