UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications

Runqing Yang [†], Shiqing Ma [‡], Haitao Xu [§], Xiangyu Zhang [¶], Yan Chen [£]

[†] *Zhejiang University*, [‡] *Rutgers University*, [§] *Arizona State University*,
[¶] *Purdue University*, [£] *Northwestern University*
[†]rainkin1993@zju.edu.cn, [‡]shiqing.ma@rutgers.edu, [§]hxu@asu.edu,
[¶]xyzhang@cs.purdue.edu, [£]ychen@northwestern.edu

***Abstract*—Existing attack investigation solutions for GUI applications suffer from a few limitations such as inaccuracy (because of the dependence explosion problem), requiring instrumentation, and providing very low visibility. Such limitations have hindered their widespread and practical deployment. In this paper, we present UIScope, a novel accurate, instrumentation-free, and visible attack investigation system for GUI applications. The core idea of UIScope is to perform causality analysis on both UI elements/events which represent users' perspective and low-level system events which provide detailed information of what happens under the hood, and then correlate system events with UI events to provide high accuracy and visibility. Long running processes are partitioned to individual UI transitions, to which low-level system events are attributed, making the results accurate. The produced graphs contain (causally related) UI elements with which users are very familiar, making them easily accessible. We deployed UIScope on 7 machines for a week, and also utilized UIScope to conduct an investigation of 6 real-world attacks. Our evaluation shows that compared to existing works, UIScope introduces neglibible overhead (less than 1% runtime overhead and 3.05 MB event logs per hour on average) while UIScope can precisely identify attack provenance while offering users thorough visibility into the attack context.**

I. Introduction

When security alerts are raised, a swift attack investigation should be conducted to determine the cause and scope of the attack so that the impact of the attack can be minimized and such attacks in the future can be prevented. Provenance tracking and causality analysis is an important technique for efficient attack investigation. Starting from a compromised system entity (e.g., file, socket, or process), investigators perform causality analysis on the collected provenance to figure out: 1) the root or origin of the entity, i.e., all the external entities (e.g., a socket connection) affecting the target entity, and 2) the causal path from the root to the entity. Such analysis facilitates identifying attack root cause, assessing damage incurred, and developing countermeasures. GUI applications (e.g., browsers), one of the most popular attack vectors [7], typically cause substantial accuracy degradation of causality analysis as they may run for a long time and process many independent tasks. In this paper, we hence focus on GUI applications.

System event (e.g., system calls) auditing is a built-in feature in mainstream operating systems and can be used for such investigation. Existing work [41], [28], [25], [30], [45], [52], [42], [37] has demonstrated their great potential, but they suffer from a few major limitations.

1) Inaccurate analysis results. In many causality analysis, when a long-running process interacts with many input and output objects, each output object will be conservatively considered causally dependent on all the preceding input objects. This is known as the *dependency explosion* problem. Such problems lead to significantly inaccurate analysis results when there are long-running processes involved and makes the investigation impossible to move forward.

2) Requiring instrumentation on end-user systems. Some approaches try to solve the dependency explosion problem using instrumentation on source code or binary. However, instrumentation is generally not practical and prohibited in real-world production environments. Firstly, most COTS software only provides executable binaries and do not provide source code. Secondly, intrusive modification of binary code can make applications and operating system unstable. Even more, it can introduce new vulnerabilities which can be leveraged by malware [11], [9], [2]. As such, Microsoft integrated the Kernel Patch Protection (KPP) into Windows to prevent patching the kernel [8]. Thirdly, the party which instruments COTS executables or operating systems has to take full responsibility for all potential accidents, no matter whether the accidents are caused by instrumentation or not. Hence instrumentation is mostly prohibited in the enterprise environment.

3) Lack of visibility. Human-perceivable attack investigation report is largely preferred and would greatly facilitate security experts to understand attacks and take necessary remedies. Some existing non-instrumentation studies employ statistical analysis [29], [41] (e.g., only considering causal edges that rarely happened in the past) to guide causal dependency graph pruning to address the inaccuracy in causality analysis. However, they just provide system level information (e.g., process id, file name) and cannot fully recover what happened from the user's perspective, which plays a vital role in the forensic analysis [53].

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24329
www.ndss-symposium.org

Our solution. We propose UIScope, a novel accurate and instrumentation-free attack investigation system which provides meaningful contextual information to enhance the visibility of forensics analysis. The basic idea of UIScope is to combine low-level causality analysis with high-level UI elements and events analysis to grain the advantages of both. On one hand, we leverage detailed low-level system events to fully recover what actually happens in the system. On the other hand, we attribute the system events to high-level UI elements and UI events to provide better visibility for attack forensics and solve the dependence explosion problem (i.e., attributing system events to individual UI elements instead of a single long-running process to avoid dependency explosion). By instrumentation-free, we mean that UIScope leverages existing built-in event logging system and does not require extra instrumentation on the end user systems.

In UIScope, there are two types of event collectors: the UI elements and events collector and the system events collector. For the UI elements and events collector, we leverage the accessibility service shipped with major operating systems, and for the system event collector, we leverage built-in audit systems such as Event Tracing for Windows (ETW). These systems are provided by OS vendors and thus usually have very low runtime and storage overhead. As such, we avoid instrumenting end user systems.

After collecting all UI events and system events, UIScope performs causality analysis on UI elements and events (through the UI event analyzer) as well as system events (the system event analyzer) to generate causal graphs for both types of logs. We devise an additional correlation analyzer, which correlates UI events and system events based on timestamp alignment and resource attribution (e.g., attributing background file accesses to the UI operation where the file resource was initialized). The details of the corresponding algorithm describing how UIScope deals with software background activities are presented in Section III-F2. Through the correlation algorithm, we partition a long-running process into individual UI transitions, making the result more accurate. Also, the final graph represents information in a way closely coupled with UI interactions which users are very familiar with, allowing to reconstruct the attack story from user's perspective and provide high visibility.

We deployed UIScope on 7 machines for a week and utilized UIScope to conduct investigation of 6 real-world attacks. Our evaluation shows that UIScope can not only accurately identify the attack path but also provide fine-grained human-comprehensible contextual semantics to users. In addition, compared to existing works, UIScope introduces negligible extra runtime overhead (less than 1%) and extra space overhead (3.05 MB event logs per hour on average). UIScope does not require any end system change or instrumentation and can be deployed as an add-on to any existing threat detection system in production environments.

In summary, we make the following contributions:

- We identify a few limitations of existing attack forensics techniques in practice and propose a novel accurate and instrumentation-free attack investigation system, UIScope, which also provides high visibility in attack forensics. It leverages both UI information in the user space and system events in the kernel space to achieve our aforementioned design goals.
- We devise a novel UI event analyzer which analyzes UI events causality, and a correlation analyzer which correlates UI events with system events to produce accurate and highly visible causal graphs.
- Based on our design, we build a prototype on Windows and evaluate on 7 machines and 6 real-world attacks. Results show that compared to existing works, UIScope introduces negligible extra runtime overhead (less than 1%) and extra space overhead (3.05 MB event logs per hour on average) while provides more accurate and highly visible attack forensics.

II. Motivation

A. Motivating Example

One day, an office clerk Bob tried to download a piece of software (`WinSCP.exe`) from a website `benign.com`. While waiting for the download to complete, he received an email informing that he had been selected as a winner for an iPad and asking him to claim the award on a website. It was actually a phishing email leading to a malicious website. The website uses the `WordPress` free host service and uses the domain name `well-known.wordpress.com`. It also leverages login detection techniques [10] to test if the user has logged into any well-known bank website. In our attack story, Bob happened to login `www.bank.com` to manage his company's bank account. Hence, the malicious website launched a clickjacking attack [5] by creating a transparent frame containing a transfer form with the receiver being the attacker's own bank account on top of the "Get iPad" button. Bob clicked the "Get iPad" button, which actually triggered a transfer from his bank account to the attack's bank account, without being noticed by Bob.

The good news was that Bob's company deployed protection techniques. The security system raised an abnormal transfer alert. Following this alert, forensics investigation began from the abnormal transfer socket communication, and tried to find how/when/where this happened and investigate if Bob was an insider attacker who stole money from the company.

B. Existing Attack Investigation Solutions

1) ***Low-level Events Causality Analysis:*** Traditional causality analysis [34], [35] tracks the lineage of system objects (e.g., files and sockets) and subjects (e.g., processes) via system events (e.g., syscalls) and analyzes the causal relations among system objects and subjects to generate causal graphs. With such graphs, investigators can perform *backward* provenance queries with the symptom (i.e., the abnormal transfer socket) to understand the root cause of the attack or *forward* provenance queries to identify the effects of the attack.

Fig. 1 (I) shows a typical dependence graph generated by traditional causality analysis for this investigation. Note that in this figure (and also the rest of the paper), we use *diamond nodes* to denote *sockets*, *box nodes* to denote *processes*, and *oval nodes* to denote *files*. Also, nodes in red represent the symptom event(s), the events which investigation starts from. Specifically, when investigators perform backward tracking to find which website is related to the abnormal transfer, they

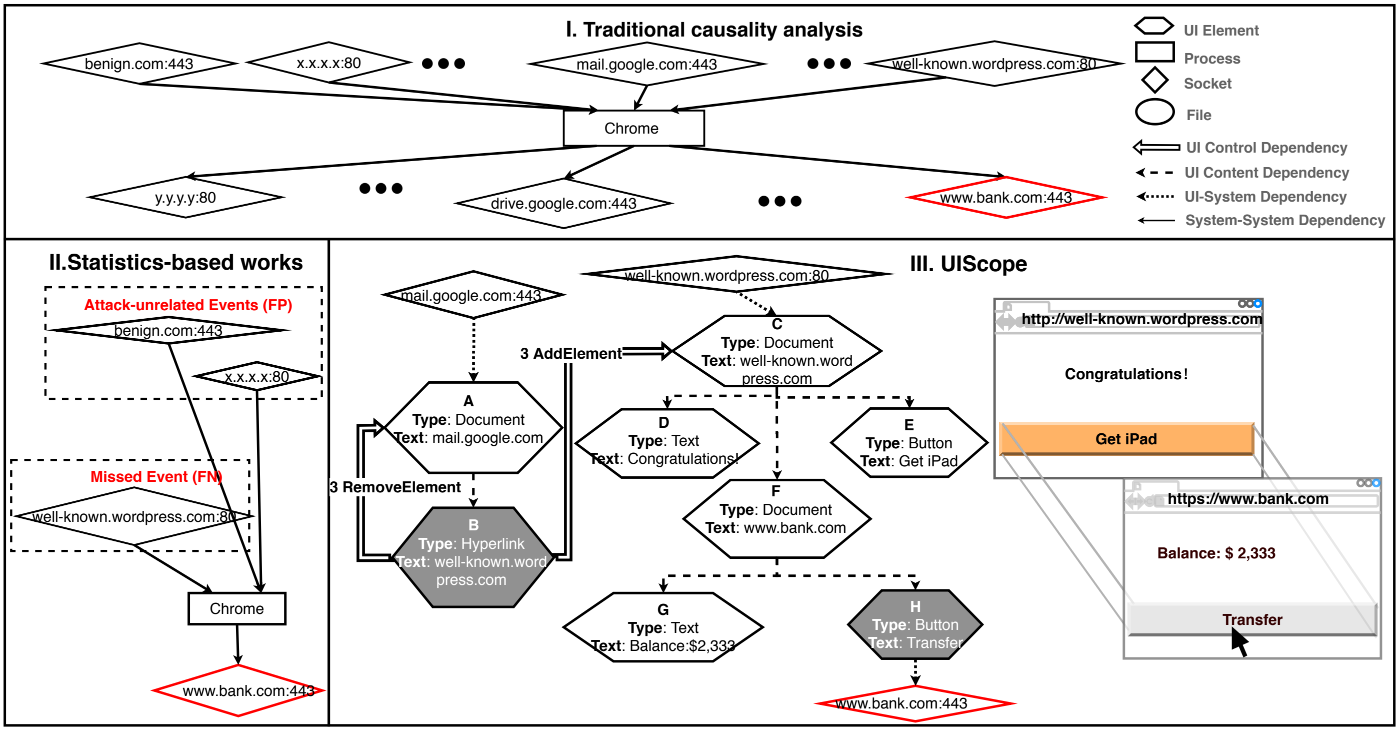

I. Traditional causality analysis
benign.com:443
x.x.x.x:80
mail.google.com:443
well-known.wordpress.com:80
Chrome
y.y.y.y:80
drive.google.com:443
www.bank.com:443
UI Element
Process
Socket
File
UI Control Dependency
UI Content Dependency
UI-System Dependency
System-System Dependency
II.Statistics-based works
Attack-unrelated Events (FP)
benign.com:443
x.x.x.x:80
Missed Event (FN)
well-known.wordpress.com:80
Chrome
www.bank.com:443
III. UIScope
mail.google.com:443
well-known.wordpress.com:80
C Type: Document Text: well-known.wordpress.com
3 AddElement
A Type: Document Text: mail.google.com
3 RemoveElement
B Type: Hyperlink Text: well-known.wordpress.com
D Type: Text Text: Congratulations!
E Type: Button Text: Get iPad
F Type: Document Text: www.bank.com
G Type: Text Text: Balance:$2,333
H Type: Button Text: Transfer
www.bank.com:443
http://well-known.wordpress.com
Congratulations!
Get iPad
https://www.bank.com
Balance: $ 2,333
Transfer

Fig. 1. Comparison between existing causality analysis solutions and our UIScope on the motivating example.

would be substantially distracted, because `Chrome` is a long running process and could have been used to visit hundreds of websites during an attack window (e.g., `mail.google.com`, `benign.com`, `drive.google.com` and many other sites in the figure), and thus the suspicious file is conservatively related to hundreds of preceding websites. This is known as the *dependence explosion problem*, which makes any further tracking attempt nearly impossible. This is a major limitation of traditional causality analysis.

Many previous works tried to address this problem via program analysis [37], [44], [45], [36]. Some of them require source code or binary level instrumentation [37], [44], [45], which is intrusive and not practical in enterprise environments, as discussed in Section I. Taint analysis [32], [51], [33] is another way to solve the dependency explosion problem. However, it causes tremendous runtime and space overhead, which makes it rarely used in production environments.

2) ***Statistics-based Graph Pruning**:* Observed that most attack-related events are abnormal and rarely occur in historical event logs of an enterprise network, PrioTracker [41] and NoDoze [29] proposed statistics-based attack investigation approaches to prioritize abnormal events and causal dependencies. They introduce quantitative metrics (i.e., frequency and topological features) to distinguish normal and abnormal events and present a pruned causal graph to investigators. PrioTracker only considers the abnormality of individual events while NoDoze takes the abnormality of event chains into consideration and thus can generate more precise causal graphs.

Fig. 1 (II) presents the causal graph generated by NoDoze for the investigation, which is more concise than the graph generated by traditional techniques (Fig. 1 (I)). However, NoDoze cannot accurately locate the IP address which is the real source of the abnormal transfer. As shown, NoDoze traces back to two benign sockets, `benign.com` and `x.x.x.x`. The reason is that visiting `well-known.wordpress.com` which is hosted by `WordPress` is a common and normal behavior (because `WordPress` is one of the largest blog host websites) while `benign.com:80` and `x.x.x.x:80` are rarely visited websites and thus deemed abnormal instead. As such, statistics based approach may produce unstable results. In other words, the qualify of the results heavily depends on the dataset used to calculate the observed distribution (e.g., which website gets more visits).

C. *Problem Statement and* UIScope

As discussed in the previous section, many low-level event causality analysis based approaches are not applicable in real world systems as they may suffer from the dependence explosion problem, causes heavy runtime overhead or requires instrumentation. Statistics based graph pruning has difficulty handling attacks leveraging popular (and benign) applications and/or websites. Furthermore, graphs generated by both approaches lack the user understandable high-level semantic information. Fig. 1 (II) shows a simplified causal graph generated by NoDoze (Note that using traditional low-level events causality analysis will introduce too many irrelevant sockets and we preclude it from our discussion). With such graphs, even if NoDoze may find the real malicious socket `well-known.wordpress.com:80` by tuning thresholds or using high quality data to calculate the observed distribution, it still cannot provide sufficient information to answer the key question in the investigation: is Bob an insider (i.e., did he intentionally initialize the transfer) or an victim of social engineering (e.g., fooled by clickjacking)? Without knowing more contextual information, it is impossible to answer this type of questions.

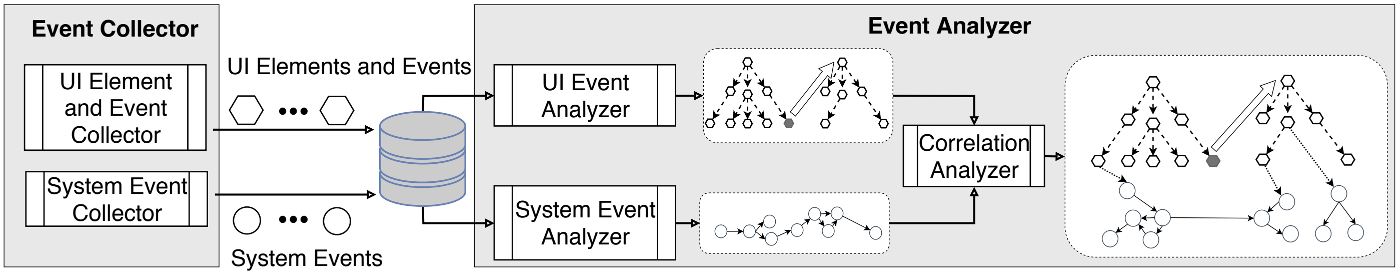


Fig. 2. UIScope System Architecture.

The aforementioned limitations motivate the following design goals for our own system:

- *Accurate:* It should produce accurate investigation results, i.e., low false positive and false negative rates.
- *Applicable:* It requires **NO** extra end system change and causes low overhead so that it is deployable in an enterprise environment.
- *Visible:* Its output should be visible to human investigators so that they can easily understand what happened with the application level context information. *Visibility* plays a vital role in the forensic analysis [53].

As far as we know, existing approaches cannot achieve all the three goals. Therefore, we propose a novel investigation system, UIScope, which combines low-level event causality analysis with high-level user interface (UI) elements/events analysis to achieve a balance of all the aforementioned goals. That is, UIScope can achieve high attribution accuracy and high visibility for GUI applications while incurring trivial overhead and requiring no instrumentation. UIScope is orthogonal to existing non-instrumentation techniques (e.g., probabilistic solutions [41], [29]). While the detailed design will be discussed in Section III, Fig. 1 (III) shows the UIScope graph for the aforementioned investigation. In this graph, we use the same shape and color to represent low-level system events as before. We also introduce a new set of nodes and edges to represent UI elements and new relations, which is defined in Section III. Specifically, hexagons represent UI elements and nodes with gray background denote UI elements operated by the user. Dashed lines show the tree structure of different UI elements pointing from the parent node to the child node, which denotes causality as well. A dotted line represents connection between a UI element and a system event and the arrow is from the action initiator. A double solid line represents connection between two UI elements and the destination is affected by the source via some user operation. For example, in Fig. 1 (III), hexagon H represents the `Transfer` button that was actually clicked by Bob; hexagon E is the `Get iPad` button which Bob intended to click, and hexagons C, D, E, F, G are Document Object Model (DOM) elements in the same page with the button. Bob first clicked the link in his email (node B), which led him to a new web page (whose UI elements rooted in C). After Bob clicking the transparent button over the `Get iPad` button (node H), a socket was created and was used to perform the unintended money transfer. The UI element nodes in this graph tell us that in the same web page, there were two clickable buttons in an identical position. If we reconstruct what was seen by Bob, we can get a graph like Fig. 1 (III), indicating that Bob was fooled by a clickjacking attack.

Additionally, Fig. 1 (III) attributes low-level system objects (e.g., detailed socket addresses and file names) to the corresponding high-level UI elements (i.e., web pages in this case), that are well partitioned and denote autonomous subexecutions. Doing so, we can accurately associate low-level system information with individual autonomous and high-level user actions. This avoids attributing all low-level events to the same process, achieving low false positive/negative rates in causality analysis, as shown by our results in Section IV-B. Details of how the attribution is done including how background events are processed will be discussed in Section III-F.

D. Threat Model

We assume that both Event Tracing for Windows (ETW) and Accessibility libraries provided by OSes (introduced in Section III) are trusted, and audit logs (i.e., system events captured with ETW and UI events by Accessibility libraries) cannot be tampered with. Attacks that can compromise these two auditing systems are beyond the scope of this study. This assumption is consistent with previous literature [29], [41], [30]. In this paper, we focus on GUI-related attacks, in which user involvement is needed to initiate/trigger an attack (e.g., phishing attacks, driven-by downloads and insider attacks). For attacks which do not entail any user interactions, we apply the same methods as previous works [29], [41], [30].

III. System Design

A. System Design Overview

The overall workflow of UIScope is shown in Fig. 2. UIScope has two major components: the event collector and the event analyzer. Specifically, the event collector consists of the UI element and event collector and the system event collector. The event analyzer contains the UI event analyzer, the system event analyzer and the correlation analyzer.

The overarching idea of UIScope is to perform both UI event causality analysis and low-level system event causality analysis independently, and then attribute groups of low-level system events to high-level UI event. In UIScope, we use UI events to deliver visibility to investigators and solve the dependency explosion problem in low-level causality analysis by attributing low-level events to high-level UI elements and events, which are well partitioned. Doing so, UIScope can generate accurate (i.e., no dependence explosion problem) and visible (i.e., through human understandable UI elements and events) results. Moreover, we do not require instrumentation or heavyweight runtime monitoring so that our technique is applicable in production systems.

B. Event Collector

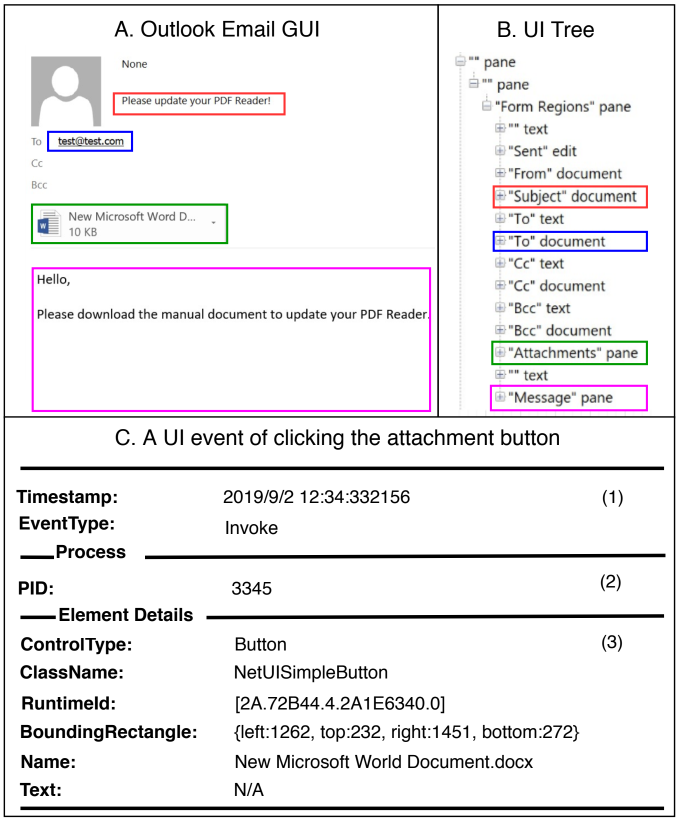
1) ***UI Element and Event Collector:*** We develop a UI element and event collector based on *Windows UI Automation* [46], to efficiently monitor interesting UI events triggered by user interactions. Windows UI Automation is an accessibility library developed by Microsoft. There are similar accessibility libraries on other platforms as well including NSAccessibility for Mac OS X [23], ATK and XAutomation for Linux [40] and so on. Enforced by the federal `IT Accessibility Laws and Policies` [27], mainstream OSes have to develop these libraries to support accessibility features. With these features, disabled people can have more control of the user interface of electronic devices, such as 1) retrieving information about UI elements (e.g., title of a window, name of a button), 2) adjusting UI elements (e.g., customizing a screen's color, zooming in and out on a web page), and 3) getting notification of changes to UI (e.g., website content loaded, a hyperlink clicked). Essentially, Windows UI Automation provides UISCOPE the capability of logging UI events triggered by any changes to UI elements.

UI Automation works by subscribing provided UI events, and UISCOPE queries the accessibility services to get the corresponding UI trees when a UI event happens. There are around 100 different UI events, falling into 5 categories, including property change, element action, structure change, global desktop change and notification [47]. Among all these events, three type of events, namely `Focus`, `Invoke`, and `SelectionItem_ElementSelected`, are used to capture common user interactions. Specifically,

- `Focus` events indicate that the focus of the user shifts from one element to another. For example, either clicking a hyperlink or yielding a new web page would trigger a `Focus` event. And at a specific time in the system, only one element can gain focus, and only that element can receive user inputs (e.g., via keyboard or mouse).
- `Invoke` events indicate when a UI element is triggered, such as clicking a button.
- `SelectionItem_ElementSelected` events indicate that an item (i.e., a group of elements) or an element is selected, such as an email in the `Outlook` or a browser tab is selected.

Besides, each UI event has a uniform format with 170 fields [48], out of which, eight fields provide wealthy semantics helpful to understand user interactions:

- `ProcessId` represents the process identifier (ID) of a UI element.
- `EventType` specifies the UI event type (e.g., `Focus`).
- `ControlType` specifies the type of the UI element which triggers the event, such as a hyperlink, a document or a tab.
- `ClassName` is the class name of the UI element assigned by software developers. Most of the time, `ClassName` together with `ControlType` can determine what kind of element triggers an event.
- `RuntimeId` is a unique identifier (ID) of a UI element, used to identify the UI element which triggers an event.
- `BoundingRectangle` is the point coordinates of a UI element's enclosing rectangle. It represents the element's position on the screen.
- `Name` is the name of UI element and is usually the same as the human-perceivable text on the screen. For example, the hypertext of a hyperlink or the title of a web page is often recorded in the `Name` field of an event.
- `Text` contains the hidden value of a UI element, such as URL of a web page or a hyperlink.



None

Please update your PDF Reader!

To test@test.com

Cc

Bcc

New Microsoft Word D...
10 KB

Hello,

Please download the manual document to update your PDF Reader.

Timestamp: 2019/9/2 12:34:332156 (1)
EventType: Invoke

Process

PID: 3345 (2)

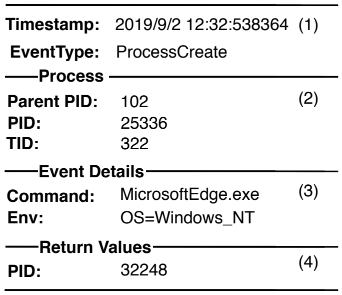
Element Details

ControlType: Button (3)
ClassName: NetUISimpleButton
RuntimeId: [2A.72B44.4.2A1E6340.0]
BoundingRectangle: {left:1262, top:232, right:1451, bottom:272}
Name: New Microsoft World Document.docx
Text: N/A

Fig. 3. Example of our collected UI information of the Outlook email composition page. (A) is the Outlook GUI which was seen by users, and (B) is the corresponding UI tree. (C) is a UI event raised by clicking the email attachment button highlighted in green in (A) and (B).

Example. Fig. 3 (A) presents an example of our collected UI information of the Outlook email composition page, in which important UI elements are marked in different colored boxes. Fig. 3 (B) shows the corresponding UI tree, and each node in the tree denotes a UI element. Our UI element and event collector will collect such information at runtime, and the UI can be reconstructed (during analysis) with such information. In Fig. 3 (C), we showcase a UI event of clicking the attachment button. As shown in the figure, we will collect event related information (C-1), process related information (C-2) and also UI element related information (C-3).

2) ***System Event Collector:*** UISCOPE uses OS built-in audit systems for system event collection. These systems are pre-installed on target systems. Without any additional configuration or optimization, they can be used in production runs. They are also of high quality and have technical support from official providers. UISCOPE leverages Event Tracing for Windows (ETW) [15] to collect system events on Windows. It allows users to collect system-level events (e.g., system calls) with negligible overhead [15]. ETW has been widely in both academia [29], [45], [25], [24], [42] and industry [16]. Similar to previous attack investigation tools [29], [41], UISCOPE only monitors security-relevant events.



```
Timestamp:   2019/9/2 12:32:538364  (1)
EventType:   ProcessCreate
——Process——————————————————————————
Parent PID:  102                    (2)
PID:         25336
TID:         322
——Event Details————————————————————
Command:     MicrosoftEdge.exe      (3)
Env:         OS=Windows_NT
——Return Values————————————————————
PID:         32248                  (4)
```

Fig. 4. Example of an ETW event.

Example. Fig. 4 shows an example ETW event entry. The first block (1) includes the basic information of the event such as the timestamp of the event (Timestamp), the event type (EventType) and so on. The second block (2) shows the process and thread triggering the event and its parent process. The third block (3) lists details of the event. For instance, for a *ProcessCreate* event, we will have the command of the created process and the environment where the created process was executed. The last block (4) has return values. In this case, it returns the child process PID value.

C. Event Analyzer Definitions

The event analyzers analyze the collected UI and system logs to derive dependencies between event entries (to construct the final causal graph). In this section, we first define a number of dependencies and then explain how they are derived.

UI Control Dependency. When a UI element change directly affects other UI elements, we say they have UI control dependencies. A UI element can affect another one directly by adding and removing operations. Thus, there are two types of UI Control Dependencies: AddElement (for adding a UI element) and RemoveElement (for removing a UI element. For example, after the user clicks a hyperlink on a web page and a new page is loaded to replace the current one. We will create a `RemoveElement` edge from the hyperlink element to the root element of the current page and an `AddElement` edge from the hyperlink element to the root element of the new page. It implies that all elements in this tree will be (directly or transitively) affected. In our paper, we use double solid lines to represent this type of dependency.

UI Content Dependency. UI elements are organized in a tree structure. A parent tree node usually represents a larger element such as a web page, and all the child nodes represent sub-elements in this web page. The affiliation relations of UI elements introduce *content dependencies*. In our paper, we use dashed lines to represent this type of dependency.

UI-System Dependency. A UI-system dependency is introduced if the UI element lead to the system event (e.g., creating a new socket for page loading in browsers) or the UI element is affected by the system event (e.g., loading data from a socket to refresh a web page). Such dependencies are critical for attributing low-level system events to execution structures enforced by UI elements (to avoid dependence explosion). In our paper, we use dotted lines to represent this type of dependency.

System-System Dependency. Previous literature [34] clearly defined dependencies between system objects (i.e., Files, Sockets) and subjects (i.e., Processes). We adapt the same definition and use the term `System-System Dependency` to represent this category of dependency in our paper. In our paper, we use solid lines to represent this type of dependency.

In addition, we use the term *initial event* to denote system object creation events.

Algorithm 1 UI Event Analyzer

Input: L_U - List of UI events in the chronological order
Output: $Graph$ - Provenance graph whose vertexes are UI elements and edges UI Control or Content Dependency.
Functions: $GetUITree(U)$ - Returns the entire UI tree related to a UI event U
$FindRootOfAddedElements(T_{cur}, T_{prev})$,
$FindRootOfRemovedElements(T_{cur}, T_{prev})$ - Returns the root element of a set of added or removed UI elements by comparing two UI trees
$Graph.addVertex(E)$ - Add a UI element to the graph if it is not included
$Graph.addEdge(E_{src}, E_{dst}, DependencyType)$ - Add a edge from E_{src} to E_{dst} if the edge does not exist and the type of edge is set by $DependencyType$
$GetChildren(E)$ - Returns a set of child nodes of UI element E in a tree
Variable: $T.root$ - the root element of the UI Tree T
$U.element$ - The UI element operated by a UI event U

```
 1: function UIEventAnalyzer($L_U$)
 2:     $U_{cur} \leftarrow null$
 3:     $T_{cur} \leftarrow null$
 4:     $U_{prev} \leftarrow L_U[0]$
 5:     $T_{prev} \leftarrow GetUITree(U_{prev})$
 6:     $AddTreeToGraph(T_{prev}.root, Graph)$
 7:     for $i = 1; i < Size(L_U); i++$ do
 8:         $U = L_U[i]$
 9:         $U_{cur} \leftarrow U$
10:         $T_{cur} \leftarrow GetUITree(U)$
11:         /* Build AddElement UI Control Dependency */
12:         $E_{add} \leftarrow FindRootOfAddedElements(T_{cur}, T_{prev})$
13:         $AddTreeToGraph(E_{add}, Graph)$
14:         $Graph.addEdge(U_{cur}.element, E_{add}, AddElement)$
15:         Set timestamp of above new added edge as the timestamp of $U_{cur}$
16:         /* Build RemoveElement UI Control Dependency */
17:         $E_{remove} \leftarrow FindRootOfRemovedElements(T_{cur}, T_{prev})$
18:         $Graph.addEdge(U_{cur}.element, E_{remove}, RemoveElement)$
19:         Set timestamp of above new added edge as the timestamp of $U_{cur}$
20:         $U_{prev} \leftarrow U_{cur}$
21:         $T_{prev} \leftarrow T_{cur}$
22:
23: function AddTreeToGraph($E_{root}, Graph$)
24:     for each $E_{child} \in GetChildren(E_{root})$ do
25:         /* Build UI Content Dependency Recursively */
26:         $Graph.addVertex(E_{root})$
27:         $Graph.addVertex(E_{child})$
28:         $Graph.addEdge(E_{root}, E_{child}, UIContentDependency)$
29:         $AddTreeToGraph(E_{child}, Graph)$
```

D. UI Event Analyzer

The UI analyzer is to find UI control dependencies and content dependencies. The detailed analysis procedure is presented in Algorithm 1. For a list of given UI events in the chronological order and also all the related UI elements organized in trees. Our goal is to compute a provenance graph, with UI elements as nodes, connected by UI control dependency and content dependency. After initializing the variables (lines 2 to 6), the algorithm starts to examine individual UI events and related UI Trees in the queue (lines 7 to 21) to build *UI Control Dependency* based on temporal association. Basically, changes in the UI Tree causally depend on the UI element operated by the previous UI event, based on the fact that there exists only one human user action at a time and that UI changes following it are responding to such action. For each event, after getting the related UI tree (line 10), it finds all the associated UI elements in the newly added or removed tree and

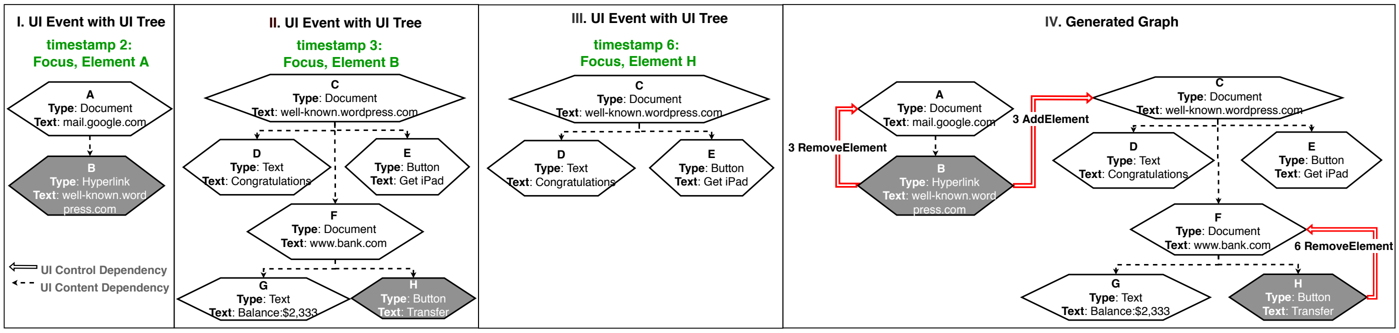


Fig. 5. A log example of UI Event Analyzer.

updates the graph accordingly (lines 11 to 19). Specifically, it introduces AddElement and RemoveElement types of control dependency to the root element of all added and removed UI elements, respectively. Note that, all added or removed elements only have one root element and we adopt the existing lowest common ancestor algorithm [17] to find the root element. Lastly, it updates U_{prev} and T_{prev} to the current UI event and the current UI tree respectively (lines 20 to 21). If UI events do not add or remove elements, it simply updates the T_{prev} and U_{prev} without adding new edges to the graph. We use the function AddTreeToGraph($E_{root}, Graph$) to add a new tree to the existing graph (lines 24 to 29). In this function, the algorithm recursively traverses the whole UI tree starting from the root element E_{root}, creates *UI Content Dependency* between parent nodes and child nodes, and adds them all to the graph.

Example. Fig. 5 presents an example of how Algorithm 1 works on the motivating example where Bob clicked a hyperlink in the phishing email and was attacked by click-jacking. Fig. 5 (I), (II), and (III) are UI events and their corresponding UI trees occurring at timestamp 2, timestamp 3, and timestamp 6, respectively. UI elements with gray background represent the ones operated by users. Fig. 5 (IV) is the generated graph by the event analyzer. At timestamp 2, the web page mail.google.com was loaded and a *Focus* event was triggered. Because this is the first UI event, UI elements A and B were added to the graph. At timestamp 3, Bob clicked the hyperlink (element B), then the previous web page mail.google.com (elements A and B) was removed from the tree and a new web page well-known.wordpress.com (elements C, D, E, F, G, H) was added. The new page was thus added to the graph, together with a RemoveElement edge from element B to the root element of the removed page (element A) and an AddElement edge from element B to the root element of the new page (element C). Note the timestamp of the edges is 3. At timestamp 6, Bob was tricked to click the Transfer button (element H) masqueraded as a Get iPad button (element E). The embedded web page www.bank.com (elements F, G, and H) was removed, thus a RemoveElement pointing from element H to the root element of the removed page (element F) was added with timestamp 6.

E. System Event Analyzer

We follow the standard causality analysis method used in previous literature [37], [44], [45], which returns a causal graph with system subjects or objects as nodes and System-System Dependencies as edges.

F. Correlation Analyzer

1) UI-System Dependency: UI events and system events capture behaviors of the same attack from two different levels: *foreground* with visibility and *background* with fine-grained information. After acquiring the UI causal graph from the UI event analyzer and the low-level event causal graph from the system event analyzer, we analyze the correlations of the two so that low-level system events can be attributed to individual UI elements (instead of the whole process) which makes the graph both visible and accurate (more accurate than using a single node to represent the whole graph like in traditional graphs, which leads to dependency explosion).

We devise a timestamp-based attribution approach based on two observations: 1) an application has only one currently focused UI tree at a certain time; 2) system events and UI events are typically triggered by the same attack behavior at the same time. That is, the two categories of events are mostly time-aligned. There are also many background events and we will discuss them in III-F2.

System events and UI events are correlated based on two mechanisms. Firstly, for a system object that is associated with a sequence of system events such as a socket with many socket read/write events, we use the creation of system object to correlate with UI events. This is known as the **initial event based correlation**. Secondly, we mainly use a timestamp based alignment to correlate general system events and UI elements/events, and this is referred as the **timestamp based correlation**. Basically, system events occurring between two UI events chronologically would be attributed to the former UI event (or UI elements added by the UI event). The details of the attribution process is presented in Algorithm 2. After initializing variables (line 2), the algorithm generates two graphs using the UI Event Analyzer and the System Event Analyzer, respectively, and add them to the result graph (lines 3 to 6). Then it starts to examine individual processes occurring in the graph (lines 7 to 15). For each process, it first checks whether the process has GUI. If so, it extracts two sub-lists of events related to the process and attributes system objects or subjects to UI elements by invoking the function $Attribute$ (lines 10 to 12).

The function $Attribute()$ first orders all system and UI events chronologically in list L_U (line 18). For each event

Algorithm 2 Correlation Analyzer

Input: L_S - A list of system events in the chronological order
L_U - A list of UI events in the chronological order
Output: $Graph$ - A complete provenance graph combining both UI and system events.
Variable: E_{active} - The current active UI element
M_{init} - A key-value map of initialized objects and active UI elements
$U.element$ - The UI element operated by a UI event U
$S.sink, S.source$ - The system object pointed to or from by a system event S
Function: $GetProcessVertexes(Graph)$ - Return all vertexes of processes
$ExtractEventsByProcess(L, process)$ - Extract events related to a specific process from a list of events L.
$AddGraphToGraph(G_{from}, G_{to})$ - add the vertexes and edges in G_{from} to G_{to} if those vertexes and edges do not exist in the G_{to}
$ReorderInChronologicalOrder(List_{one}, List_{two})$ - merge two lists into a list in the chronological order
$FindRootOfAddedElements(E, Graph)$ - check whether there is AddElement dependency pointing from the element E, if yes, return the sink element of the dependency, otherwise return $null$.

```
 1: function CORRELATE($L_S$, $L_U$)
 2:     $Graph \leftarrow$ a new empty graph
 3:     $G_{system} \leftarrow SystemEventAnalyzer(L_S)$
 4:     $G_{ui} \leftarrow UIEventAnalyzer(L_U)$
 5:     $AddGraphToGraph(G_{system}, Graph)$
 6:     $AddGraphToGraph(G_{ui}, Graph)$
 7:     for each $process \in GetProcessVertexes(G_{system})$ do
 8:         if $HasGUI(process)$ then
 9:             /* For GUI applications*/
10:             $SubL_{system} \leftarrow ExtractEventsByProcess(L_S, process)$
11:             $SubL_{ui} \leftarrow ExtractEventsByProcess(L_U, process)$
12:             $Attribute(process, SubL_{system}, SubL_{ui}, Graph)$
13:         else
14:             /* For non-GUI applications, we follow traditional methods*/
15:             continue
16:
17: function ATTRIBUTE($process, L_{system}, L_{ui}, Graph$)
18:     $L_{event} \leftarrow ReorderInChronologicalOrder(L_{ui}, L_{system})$
19:     for $i = 0; i < Size(L_{event}); i++$ do
20:         $event \leftarrow L_{event}[i]$
21:         if $IsUIEvent(event)$ then
22:             $U \leftarrow event$
23:             $E_{add} \leftarrow FindRootOfAddedElements(U.element, Graph)$
24:             if $E_{add} \neq null$ then
25:                 /* Activate the root of the new added elements */
26:                 $E_{active} \leftarrow E_{add}$
27:             else
28:                 /* Activate the operated element if no new element occurs */
29:                 $E_{active} \leftarrow U.element$
30:         else if $IsSystemEvent(event)$ then
31:             $S \leftarrow event$
32:             if $IsInitialEvent(S)$ then
33:                 /* Timestamp-based approach */
34:                 $M_{init}.put(S.sink, E_{active})$
35:                 $Graph.addEdge(E_{active}, S.sink, UISystemDependency)$
36:             else
37:                 /* Initial event-based approach */
38:                 $E_{init} \leftarrow M_{init}.get(S.object)$
39:                 if $S.source = proc$ then
40:                     $Graph.addEdge(E_{init}, S.sink, UISystemDependency)$
41:                 else
42:                     $Graph.addEdge(S.source, E_{init}, UISystemDependency)$
```

in the list, it first checks the event type. If it is a UI event, it checks whether the UI event causes new elements in the UI tree. If so, it marks the root element of newly added UI elements as the current active element (lines 24 to 26). Otherwise, it marks the element operated by the current UI event as the active one (lines 27 to 29). Basically, system events occurring between two active elements chronologically will be attributed to the former element. If the event is a system event, it first tests whether the event is an initial event (line 32). If so, it records the relationship between the initialized resource and the current active element in the map M_{init} (line 34). Then it adds a new *UI-System Dependency* edge pointing from the active element to the sink (line 35). Otherwise, it retrieves the UI element related to the operated resource from previous records in M_{init} and conducts the same operations as before (lines 38 to 42). Note that the direction of the newly created edge is the same as the one of the system events. For instance, the `FileRead` event creates an edge pointing from a file object to a UI element while the `FileWrite` event creates an edge pointing from a UI element to a file object. The essence of lines 38 to 42 is as follows. We observe that the usage of a system object (e.g., socket read) may not be in the same time window of a UI event, rendering the timestamp-based alignment (lines 33-35) ineffective, while its initialization (e.g., socket creation) can be correctly aligned with the trigger of the UI event (e.g., button clicking) in most cases. Thus, for a (background) system event related to some system object, we trace back to the time when the system object was initialized and attribute the event to the active UI element at that moment.

*2) **Background Activities:*** One may question if our technique (based on timestamps and system object initialization) provides a sound solution for background activities. In the following, we discuss the possible background behaviors of popular applications and argue that our technique is highly effective in practice, which is also demonstrated by our experiment in Section IV-B.

GUI programs are mostly driven by keyboard and mouse inputs. The execution trace of such a program is dominated by event processing loops and the system events that are directly or indirectly triggered by the loop body. UISCOPE captures keyboard and mouse inputs by monitoring UI events, and then uses timestamp-based correlation and initial event-based correlation to attribute system events to UI events. We classify GUI applications into two categories: *static* and *dynamic* based on if there are activities triggered in the background. For static applications(i.e., no background activity), UI events and corresponding system events are triggered synchronously and UISCOPE works well. For example, there are no background activities in `Notepad PlusPlus` when there are no user inputs. For dynamic applications (i.e., with background activities), using time-based correlation is not enough. For example, `Chrome` can download videos or other files in the background while the user is browsing in the foreground. Browser is the most common and complex dynamic application type. In the following, we will use browsers as an example to demonstrate how UISCOPE handles background activities.

The accuracy of UISCOPE for browsers is affected by 1) the types of visited websites, 2) the duration of web sessions, 3) the number of websites with background activities. We use Chrome to monitor the behaviors of 500 popular websites in an hour after being loaded. Those websites are selected from Alexa top websites [1]. Foreground activities will trigger new UI events that will be captured by our system, and UISCOPE has high attribution accuracy. Thus, we focused on monitoring website behaviors that are left in the background without user interactions. Specifically, the GUI automation tool [4] first triggers typical behaviors of a website (e.g., playing video on `www.youtube.com`) to simulate initial user activities when the website was visited. Then it does not interact with the website anymore to simulate the scenario in which the website has been put in the background. Furthermore, we only consider network events because network events in the background are much more complex and diverse than file events. For instance, the cache and cookie of all websites loaded by Chrome are stored in several deterministic files or folders (e.g., an `SQLite` database for cookies). Furthermore, our preliminary results

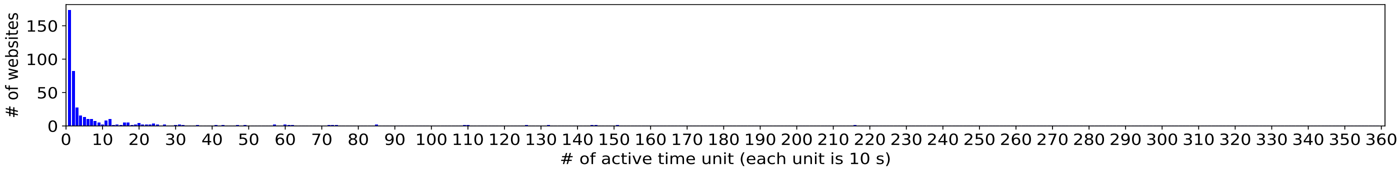


Fig. 6. The number of active time units is small for most websites.

on 20 minutes running of 20 background web pages show that 84% of total file events are related to 10 files, several of which are the aforementioned cache and cookie files. It indicates that file access in Chrome is simple and typical. Finally, 436 websites were successfully monitored and used in this experiment are listed in Table VI. After analyzing the data, we make a few important observations.

Observation 1: Most new website activities are narrowed in a short period of time and activities in the background are not common. We monitor website activities every 10 seconds as a unit and count the total number of units the website initializes at least one new socket. The result is shown in Fig. 6. The Y-axis in the graph represents the number of websites and the X-axis represent the number of active units. For example, the first bar denotes that 173 websites are active in only one unit (10 seconds). From the graph, we know that 71.11% of the 436 websites finish all the work within 1 minute and 89.44% of websites are idle for at least 55 minutes during the whole time (1 hour).

Observation 2: Most socket sessions are initialized during web page loading. Fig. 7 shows the average number of initialized socket sessions within 60 seconds after a website is visited. For such sockets, we observe that around 87.13% of all socket sessions are initialized within 10 seconds and 94.11% socket sessions are initialized within 20 second. Together with the analysis results in observation 1, it verifies the usefulness of our timestamp-based event attribution approach (Section III-F).

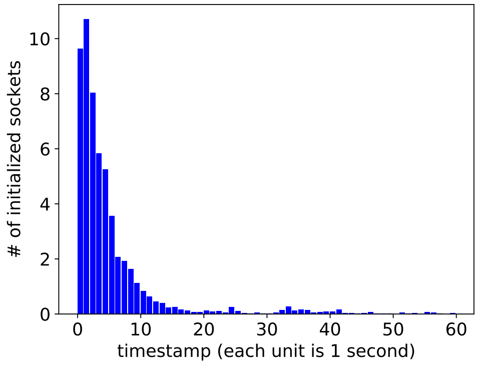


Fig. 7. The number of initialized socket sessions within the first 60 second.

Observation 3: Background activities are usually repeated behaviors. We analyzed the socket connection destinations of all sockets, including both existing sockets and new sockets, and the results are shown in Fig. 8. Blue bars are the number of both types of sockets and red bars are the number of newly created sockets. The X-axis is time, and one hour period is divided to 360 units. As we can see, there are not many new sockets after the pages are loaded, which implies that websites in the background connect to a limited set of domain names repeatedly. To further verify our finding, we manually check with source code of some of these sites. Fig. 9 shows 4 websites that have the maximum number of background activities. The Y-axis represents the top 10 frequent top-level domain names and X-axis shows the number of socket sessions that are related to a certain domain name. As we can see, most socket connections are related to a limited number of top-level domain names, indicating that background behaviors are largely repeated connections. This allows UIScope to use pattern based filters to remove such background activities.

Adaptive Adversary. In an adaptive attack scenario in which the attacker is aware of the presence of UIScope, the attacker may intentionally use delayed background activities that do not have the resources initialized when the UI event occurs (e.g., using a timer to postpone the creation of a socket to download a payload). Such attacks may lead to incorrect attribution of low-level events to UI operations. But we argue that 1) this is a common unsolved challenge for all existing attack investigation systems (e.g., [34], [35], [41]). 2) Practically, it is very difficult to perform such attacks, and many popular attack vectors cannot be delayed. For example, about 71% attacks use phishing emails for initial compromise [7], and the adversary is not able to do delay actions in this scenario.

Example. Fig. 10 presents an example of how Algorithm 2 works on the motivating example. Fig. 10 (I) and (II) show the graphs generated by the UI Event Analyzer and the System Event Analyzer. Fig. 10 (III) shows the final output graph. We use the logic time in this case to denote the order of all the events. Elements B and H were clicked at timestamps 3 and 6 respectively, which lead to two sockets being initialized at timestamps 4 and 7. According to the algorithm, the root (element C) of newly added elements and the operated element (element H) were marked as active at timestamps 3 and 6, respectively. Thus, the socket events are attributed to elements C and H, respectively. System events occurring at timestamps 5 and 9 are non-initial events and we attribute them to elements C and H through the tracing-back to initial system object method. The red dotted line in Fig. 10 represents UI-System Dependency defined in Section III-C. Note that `benign.com:80` was initialized before those two UI events for downloading the software `WinSCP.exe`, thus it was attributed to previous UI elements. Furthermore, the non-initial system event related to `benign.com:80` occurring at time 8 was attributed to the previous element to which the socket creation belongs to. *Observe that by separating a process to different autonomous UI trees and attributing low-level system events to these trees, our technique achieves the effect of partitioning a long-running execution to autonomous units, which is the key to avoiding dependence explosion.*

IV. Evaluation

We evaluate UIScope by answering following questions.

- **Q1:** How much runtime and space overhead does UIScope incur in production environments? (Section IV-A)

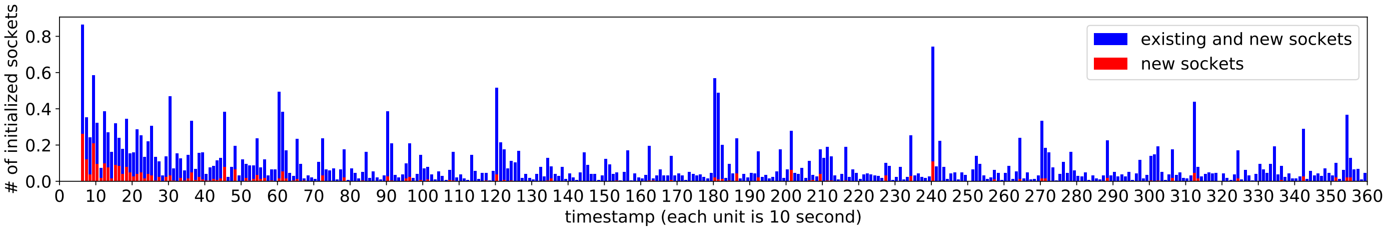


Fig. 8. The number of initialized socket for each time slice starting from 1 minute to 1 hour.

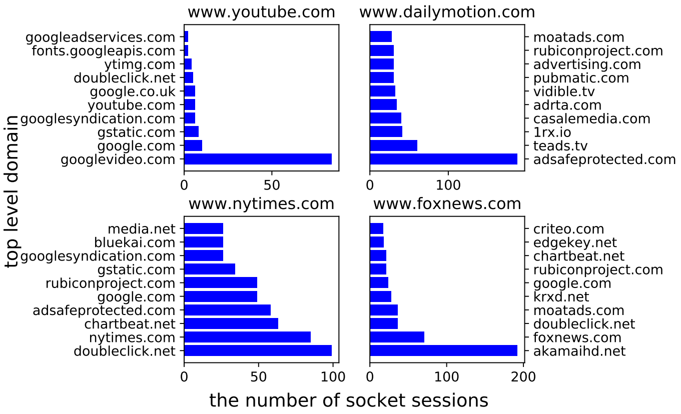


Fig. 9. Long-term active website examples.

- **Q2:** How accurately can UIScope cohere UI elements and System objects/subjects? (Section IV-B)
- **Q3:** How effective is UIScope in conducting real-world attack investigation and how does it compare to the state-of-the-art? (Section IV-C)

Experiment Setup. We deploy UIScope on 7 Windows computers and run experiments for a week to collect UI events and system events triggered. Note that we do not require those 7 users to perform any specific actions, and our purpose is to collect event traces generated by their daily usage. In addition, we simulate six real-world attacks listed in Table V to evaluate the effectiveness of UIScope in attack investigation.

TABLE I. The Runtime Overhead of the UI Collector

App	Native (ms)	UI Collector (ms)	Overhead
Notepad	22882	22909	0.12%
Notepad PlusPlus	21458	21502	0.20%
Sublime Text	30386	30415	0.10%
Explorer	31090	31174	0.27%
Paint	28551	28713	0.57%
Snipping Tool	13785	13791	0.05%
Chrome	48511	48836	0.67%
Edge	50832	51184	0.69%
WinScp	32080	32186	0.33%
FileZilla	34511	41079	19.03%
Outlook	33138	33404	0.80%
Skype	29085	29336	0.86%
Adobe Reader	35356	35581	0.63%
Foxit Reader	36744	37057	0.85%

A. Performance Overhead

UIScope includes two event logging tools, the UI collector and the system event collector, for trace collection on end hosts. System event traces are deemed to be the major data source for most existing attack investigation systems. UIScope uses the Windows built-in audit system ETW as its system event collector. ETW has been evaluated to be quite lightweight, only imposing 0.4% ∼ 2.5% runtime overhead [42]. The space overhead caused by ETW mainly depends on what event types ETW is configured to capture. We use ETW to monitor the same system event types (i.e., only the security-relevant types) as existing works [38], [57], [50], [28], [31], and thus the space overhead by ETW in UIScope is around 210 MB per week after deploying their system, which is reasonable. Therefore, we focus on evaluating the run-time overhead and space overhead introduced by the UI collector.

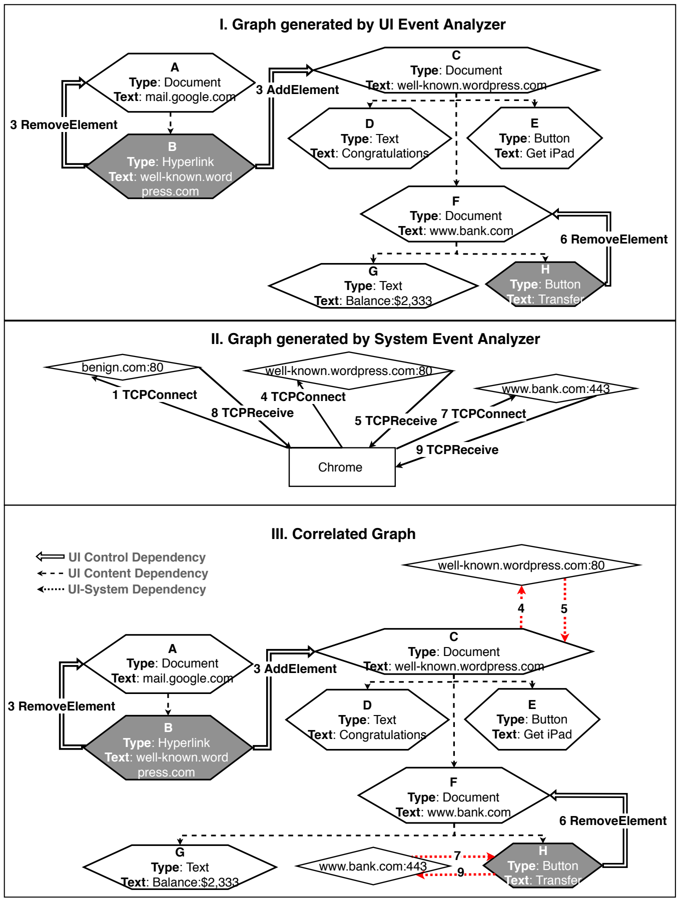


Fig. 10. A log example of correlation analyzer.

*1) **Runtime overhead:*** We evaluate the runtime overhead of UI collector in UIScope for 14 different GUI applications. Firstly, we collect a lot of application traces from real-world scenarios. Then, we automate these workloads in two different environments, i.e., with and without UIScope collecting UI events and trees. Table I shows the results. Column 2 represents

TABLE II. SPACE OVERHEAD EVALUATION RESULTS

	S-0	S-1	S-2	S-3	S-4	S-5	S-6	Total
CPU	I5-9400	I7-6500U	I7-8700	I7-8700	I7-7500U	I7-8550U	I5-7400	N/A
RAM (GB)	8	8	16	16	16	16	8	N/A
Windows Version	7	10	10	10	10	10	10	N/A
Duration (hours)	16.7	108.7	95.4	54.6	37.4	28.7	121.9	463.4
# of UI Events/Trees	1,131	32,824	18,687	34,010	22,766	8,704	354,907	473,029
Size (MB)	14.84	251.44	55.18	179.86	81.16	23.78	753.35	1416.62

the average running time of running on native environment, column 3 represents the average running time of the same user interaction with UI collector, and the column 4 is the runtime overhead. The table shows that the UI collector introduces negligible (less than 1%) runtime overhead for almost all applications. UI collector introduces 19.03% overhead for `FileZilla`. The reason is that the GUI of `FileZilla` has its own customized UI Automation APIs [13], which issues larger overhead.

*2) **Space overhead**:* Table II presents the details about the collected UI event logs on each host. For each user machine, Table II lists the hardware configuration (i.e., CPU and RAM), Windows OS installed, duration of running the event collectors, number of UI events triggered by daily user behavior, and storage space for the UI events. Note that the users are free to enable or disable UISCOPE, so the duration of event logging varies from user to user.

In summary, being deployed on 7 user machines for a week (with a total active duration of 463.4 hours), UISCOPE collected 1416.62 MB UI log including 473,029 UI events and the corresponding UI trees. On average, the UI collector component on one machine generates 3.05 MB event logs per hour. Hence, the space overhead is negligible, compared to system logging.

B. Accuracy of Events Correlation

1) Static Applications: To evaluate the accuracy of event correlation, it is necessary to obtain the ground truth. We did the following experiments with 12 popular applications, including editor software (`Notepad PlusPlus`, `Sublime Text`, and `Notepad`), communication software (`Outlook` and `Skype`), file transfer software (`WinSCP` and `FileZilla`), PDF Reader (`Adobe Reader` and `Foxit Reader`), and miscellaneous software (`Explorer`, `Snipping Tool`, and `Paint`). These static applications introduce no or negligible background activities (e.g., checking new versions after opening the software).

For each application, we use a UI automation tool [54] to trigger a typical application specific behavior and collect system events, e.g., file opening and saving for `Notepad PlusPlus`. We repeat this process three times in isolated environments and extract a common set of system events from these three runs. The pairings of the used UI events and the corresponding common set of system events are considered our ground truth of one behavior.

For each application, we performed its specific behavior 10 times with different settings (e.g., opening 10 different files), and obtain the ground truth for the 10 instances. Next, for each application, we trigger all the 10 behavior instances, collect the generated system events, and use UISCOPE to perform the correlation. Then we compare the results with the ground truth to evaluate UISCOPE. As mentioned in Section III-F, we develop two attribution methods: *timestamp-based* and *initial event-based*. We apply different combinations of these attribution methods to the collected traces. Table III shows the attribution accuracy results on the 12 popular applications under different operations. The last row shows the average value of True Positive Rate (TPR) and False Positive Rate (FPR) for all the 12 applications.

TABLE III. ATTRIBUTION ACCURACY OF STATIC APPLICATIONS

App	Operation	Timestamp		Timestamp + Initial Event	
		TPR	FPR	TPR	FPR
Notepad	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
Notepad PlusPlus	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
Sublime Text	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
Explorer	Delete File	100	0	100	0
	Rename	100	0	100	0
	Copy Paste	100	0	100	0
Paint	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
SnippingTool	New	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
WinScp	Login	100	0	100	0
	Download	81.32	2.21	100	0
	Upload	90.23	1.13	100	0
FileZilla	Login	100	0	100	0
	Download	85.2	1.63	100	0
	Upload	80.1	2.17	100	0
Outlook	Open Email	81.4	2.06	99.2	0.07
	Upload	100	0	100	0
	Download	100	0	100	0
Skype	Upload	100	0	100	0
	Download	91.37	1.05	100	0
Adobe Reader	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
Foxit Reader	Open	100	0	100	0
	Save	100	0	100	0
	Save as	100	0	100	0
Average	-	**97.41%**	**0.29%**	**99.97%**	**0.002%**

We can see that on average 97.41% system events can be attributed to the correct UI events only based on timestamps, which is consistent with our observation in Section III-F2.

TABLE IV. ATTRIBUTION ACCURACY OF DYNAMIC APPLICATIONS

NumberOfWebsites	StayTime	Timestamp			Timestamp + Initial Event		
		TPR	FPR	# of unique domain of FP per website	TPR	FPR	# of unique domain of FP per website
10	10	88.57%	1.26%	13.06	94.20%	0.64%	2.69
10	20	90.83%	1.01%	13.51	95.52%	0.49%	2.23
10	30	90.55%	1.04%	15.49	96.22%	0.42%	2.34
20	10	85.58%	0.75%	18.97	92.53%	0.39%	3.66
20	20	86.68%	0.71%	22.38	93.54%	0.33%	3.42
20	30	85.61%	0.75%	25.27	93.47%	0.34%	3.69
30	10	83.27%	0.57%	23.82	91.48%	0.29%	4.37
30	20	84.05%	0.54%	28.04	92.19%	0.26%	4.41
30	30	77.98%	0.75%	30.74	86.55%	0.46%	4.98

When we apply both methods at the same time, the average TPR improves from 97.41% to 99.97%, and the average FPR reduced from 0.29% to 0.002%. Overall, UISCOPE achieves high attribution accuracy (99.97%) and negligible false positives. A few events were missing for the action of opening emails with `Outlook`, and it is because `Outlook` supports HTML emails that can load resources dynamically in the background. `WinScp` and `FileZilla` have relatively small TPR under only the timestamp-based method because file downloads/uploads happen in parallel. With two attribution methods enabled, UISCOPE can still get 100% accuracy.

2) Dynamic Application: We evaluate UISCOPE on `Chrome` which is the most complex dynamic application. As mentioned in Section III-F2, the accuracy of UISCOPE for browsers is affected by 1) the types of visited websites, 2) the duration of web sessions, 3) the number of websites with background activities. We evaluated UISCOPE on `Chrome` over these three variables and summarized the results in Table IV. To calculate the TPR and FPR, we use the same method in Section III-F2 to obtain the ground truth for 463 websites listed in Table VI.

In this experiment, we randomly select a given number of websites (**NumberOfWebsites**, column 1) from 463 website and load them one by one. The interval of loading two different web pages is a constant number (**StayTime**, column 2). Then we apply UISCOPE to the collected trace and calculate the TPR and FPR by comparing with the collected ground truth data. To evaluate the significance of our two approaches (timestamp-based and initial-event based), we apply the timestamp based approach (columns 3 to 5) and the timestamp plus initialization based method (columns 6 to 8). Besides FPR and TPR, we calculate the average number of unique domains involved in the false positives of each tested website (columns 5 and 8). In order to cover more types of websites, we repeat the above process 1000 times and calculate average TPR and FPR for each setting, and Table IV shows the average value for each setting in each row.

From the table, we can see that 1) timestamp based approach could correctly attribute most events and initial-event based approach could further increase the TPR and decrease the FPR, which conforms to the observation 2 mentioned in Section III-F2. That is, most socket sessions are initialized during web page loading. 2) increasing NumberOfWebsites decreases the accuracy because with more background websites, the more false dependencies would be introduced to the foreground website.

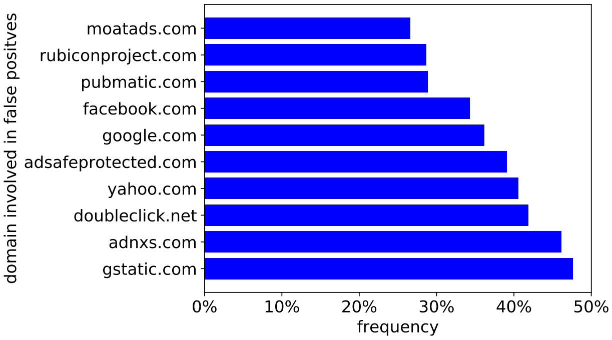


Fig. 11. Frequency of domains occurring in the false positives.

Although UISCOPE cannot handle newly created socket in the background, the number of unique domains of false positives per website is relatively small (2.23 ∼ 4.98). We further explore those false positives of each website and find that they are related to many common domains, which conforms to the observation 3 mentioned in section III-F2. Fig. 11 shows the top 10 common attributed domains by background communication. Y-axis represents domains and X-axis represents the frequency of wrong attribution. From the figure, we can see that these domains are related to advertisement (`moatads.com`, `rubiconproject.com`, `pubmatic.com`, `adnxs.com`, `adsafeprotected.com`, and `doubleclick.net`), Google services (`gstatic.com`, `google.com`) and social media (`facebook.com` and `yahoo.com`). For those frequent domains, we can remove them to further reduce the FPR. Furthermore, we evaluate `Edge` and the results show that `Edge` has similar accuracy to `Chrome`. Specifically, our system achieves 94.8% TPR and 0.51% FPR when **NumberOfWebsites** is 10 and **StayTime** is 20, and 92.7% TPR and 0.25% FPR when **NumberOfWebsites** is 30 and **StayTime** is 20.

Lastly, we want to point out that UISCOPE correlates events to prune unnecessarily large investigation graphs. Thus in this context, FP cases mean that a portion of the graph cannot be pruned by UISCOPE .

C. Attack Investigation

We demonstrate the effectiveness of UISCOPE in attack provenance tracking by applying it to 6 real-world attacks, including Phishing email [20], Remote Code Execution [55], MS Office Macro Attack [56], Credential-based Attack [22], Watering Hole Attack [18], and Insider Attack [21]. In the following, we will use the Remote Code Execution attack to

TABLE V. ATTACK INVESTIGATION SUMMARY ON THE REST 5 REAL-WORLD ATTACKS

Attacks	Short Description	Root cause by UISCOPE
Phishing email [20]	Motivating example discussed in Section II-A.	✓
MS Office Macro Attack [56]	An malicious document was downloaded and executed as an `Outlook` attachment and the enclosed macro was triggered by Excel to perform malicious behaviors.	✓
Watering Hole Attack [18]	An malicious file was uploaded to a popular forum. A victim visited the forum through Google search, and downloaded and executed the malicious file.	✓
Insider Attack [21]	An insider attacker downloaded sensitive files from a FTP server and compress and send out such files through Outlook.	✓
Credential-based Attack [22]	An attacker accessed the machine with the stolen VNC credential and transfer bank money through passwords automatically saved by Chrome.	✓

present a case study and other five attacks are summarized in Table V. The first column shows the attack name and reference. The second column summarizes the attack scenario and the last column indicates if UISCOPE can find the root cause of the attack. And we can see that, UISCOPE is capable of finding all root causes of different types of attacks.

1) Case Study: Remote Code Execution: Remote Code Execution (RCE) is a vulnerability that can provide an attacker with the ability to execute malicious code and take complete control of an affected host, no matter where the host is geographically located.

Scenario. In this attack, UISCOPE and a threat detector have been installed and enabled on a user's machine. The user accidentally navigates to a malicious `Flappy Bird` game website with the browser `Edge` by clicking on a hyperlink returned from `Google Search`. The website asks the user to press the 'enter' key to control the bird. The website leverages CVE-2018-8495 [55] to perform an attack. Once the `enter` key is held, the website will invoke a pop-up window asking if the user wants to start, and as soon as the `enter` key is released, a malicious `PowerShell` script gets executed on the victim's machine. While the user is thinking he is playing the game by using the `enter` key, he actually has been tricked to execute malicious code on his own machine.

Threat alert. Soon, the threat detector installed detects malicious `PowerShell` script running and thus raises an alert.

Investigation. An incident investigator starts investigation with our tool. With the given threat alert and the audit logs (UI events and system events) collected, UISCOPE could efficiently yield a semantics-rich human-comprehensible causal graph shown in Fig. 12. By provenance tracking in the graph, the investigator can quickly find that the malicious `PowerShell` script was from the website `http://FakeWebsite.com`, which was opened by clicking a link return by `google.com` (element B). The investigator then examines the `FakeWebsite` and finds that a suspicious hyperlink (element D) contains suspicious keywords (i.e., `PowerShell` and `vbs`) in its URL string. Close scrutiny of the suspicious hyperlink would reveal that the hyperlink exploits an Edge vulnerability CVE-2018-8495 [55] to execute a vulnerable benign VBS program `xxx.vbs` which further allows the attacker to execute any `PowerShell` code remotely to have full control over the victim's machine. However, the vulnerability would open a pop-up window to explicitly ask for user's consent. No user would be fooled into clicking 'OK' and run the program. Then the investigator looks into other behaviors of `FakeWebsite` and finds that after the website was loaded, a pop-up window (element F) with the text "How do you want to open it" showed up, and an `OK` button (element H) in the window was focused by default. And the website asks the user to press the enter key to control the game character (element E), which causes the `OK` button to be pressed. This explains how the attacker acquired the permission.

This case demonstrates that UISCOPE can not only assist identifying the attack provenance but also provide fine-grained human-comprehensible contextual semantics to users. Comparatively, Fig. 13 shows the graph generated by NoDoze. Although NoDoze contains the malicious socket `fakewebsite.com:443` in the graph because `fakewebsite.com` was rarely visited, it misses the `google.com` which is frequently visited and it cannot provide context information to understand the attack or prevent similar attacks from happening again.

V. DISCUSSION

Accuracy. UIScope works well for static GUI applications (with almost 100% accuracy) in which UI events and corresponding system events are triggered synchronously; and UIScope obtains around 90% accuracy for dynamic GUI applications (e.g., Chrome) in which system events are triggered asynchronously. If the initialization of system events and UI events do not appear close together in time, our system could result in wrong attributions. For instance, the advertisement JS code embedded in websites could use `setTimeout` API to update pictures after a certain time in the background. To attribute such system events accurately, existing solutions [39] instrument applications to track fine-grained events while UIScope introduces around 10% wrong attributions as a trade-off to avoid instrumentation. We believe UIScope is practical in real-world scenarios for a few reasons reasons: 1) more than 71% attacks use phishing emails for initial compromise [7] and the system events and UI events are typically triggered very closely in time in these scenarios; 2) UISCOPE essentially performs graph pruning, and FP cases mean that UISCOPE is not able to remove some nodes in the graph, which does not cause fatal problems in invetigation. UISCOPE filters out over 99% false positives and makes attack investigation quite effective, compared with traditional non-intrusive work [34]; 3) UIScope is orthogonal to other existing instrumentation-free techniques. We could leverage probabilistic solutions [29] to complement UIScopes results.

Privacy. Information collected by UISCOPE may raise potential privacy issues. We argue that 1) UISCOPE targets an enterprise environment in which employers have the right to monitor the computers owned by them in United States according to the law [6]; 2) Sensitive information such as passwords cannot be retrieved due to security policy encoded in the Accessibility framework; 3) There exists a lot of work (e.g., data masking) that can be used for protecting sensitive information, which can be adopted to protect users' privacy.

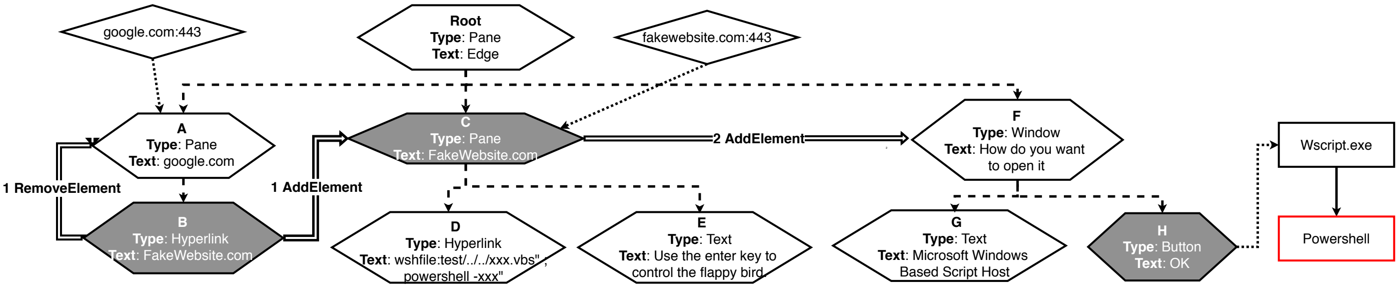


Fig. 12. Causality Graph generated by UIScope for the RCE scenario.

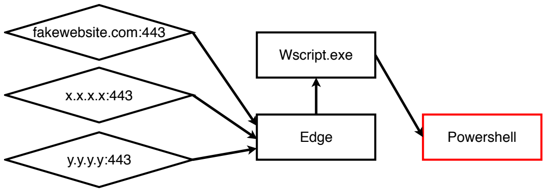


Fig. 13. Causality Graph generated by NoDoze for the RCE scenario.

Technical feasibility of UI event logging. The federal law [27] requires all applications used by U.S. Federal agencies to support accessibility, and most popular applications and OSes support such UI event logging. Considering one in four US adults living with a disability [3], accessibility support is an inevitable trend. For applications not built with the underlying system GUI APIs, Windows has provided an interface [13] to help them support accessibility on the Windows platform. Popular third-party GUI applications/frameworks (e.g., Qt and Chrome) have leveraged this interface to support accessibility.

VI. Related Works

Along with the previous works discussed in Section II-B, there exist other studies that are related to UIScope.

Attack Investigation. For Web-based attacks, JSgraph [39] is able to track and reconstruct fine-grained details about the JS code. For our motivation example, JSgraph could provide details about how the website tests if a user has logged into any well-known bank website, how it creates a transparent frame, and which JS file performs the attack, which cannot be captured by UIScope. However, JSgraph only works for Chrome and needs expensive user efforts to instrument source code while UIScope is a generic and instrumentation-free solution. For non-instrumentation techniques, HOLMES [49] focuses on detecting APT attacks and providing high-level APT stages (e.g., initial compromise, foothold establishment, and privilege escalation) that summarize the attackers action specific to the APT life cycle, while UIScope is a generic solution which attributes system events to UI elements to provide high-level context information. NoDoze [29] and PrioTracker [41] leverage statistical analysis to guide causal dependency graph pruning to address the inaccuracy in causality analysis. However, those techniques would not work on UI events because there would not be a way to differentiate anomalous from non-anomalous UI events. Furthermore, all those works only rely on system events which cannot provide visible contextual information like UIScope.

Log Reduction. There exists a line of work [38], [57], [50], [28], [31], [43] focusing on reducing the log size of system events while preserving the dependency for provenance tracking. UIScope collects the same types of system events as those works and our system only reply on must-have fields of system events (e.g., timestamp), thus UIScope is orthogonal to these approaches and can incorporate those them to reduce the storage of system events.

Other Digital Forensic Tools. Many other digital forensic tools focus on analyzing digital artifacts (e.g., memory [14] and disk [12]) left in computers after an attack happened. Then security analysts manually analyze and identify causality-related events (e.g., timeline analysis [19], [26]). Comparatively, UIScope starts to monitor the system before attacks and automatically correlate system and UI events to construct a dependency graph for provenance tracking. Some forensic tools could extract high-level semantics by analyzing application-specific logs, e.g., email forensic tools. However, those tools highly rely on the format of application logs. If logs are stored with an unknown format, those tools cannot works. Comparatively, UIScope supports capture high-level semantics of all GUI applications by monitoring UI events.

VII. Conclusion

We develop UIScope, an accurate and instrumentation-free attack investigation system with high visibility. UIScope highlights the critical role of user interaction with the system through GUI in partitioning system operations and tracing the provenance of attack. By leveraging user-space user interaction logs to complement kernel-level system events with human-perceivable contextual semantics, UIScope is able to provide accurate and visible investigation results, even to ordinary users who are not tech-savvy. UIScope is lightweight and efficient, incurring negligible performance overhead. UIScope is applicable in production environments, requiring no end system change or instrumentation. Our evaluation shows that UIScope can efficiently and precisely identify the provenance of real-world attacks.

Acknowledgment

The authors would like to thank anonymous reviewers and our shepherd, Prof. Adam Bates, for their feedback in finalizing this paper. We would also like to thank Xue Leng and Jiuyuan Wang for informative discussions on the submitted manuscript. This work is supported, in part, by NSFC under U1936215, DARPA under FA8650-15-C-7562, NSF under 1748764, 1901242 and 1910300, ONR under N000141410468 and N000141712947, and Sandia National Lab under award 1701331. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] "Alexa top websites," https://bit.ly/2N1rZPi , accessed on 2019-09-10.

[2] "Captain hook:pirating avs to bypass exploit mitigations," https://goo.gl/zVyuAL, accessed on 2019-09-10.

[3] "Cdc: 1 in 4 us adults live with a disability," http://bit.ly/2QWsrAU , accessed on 2019-09-10.

[4] "Chromedriver - webdriver for chrome," https://bit.ly/35wPgiu, accessed on 2019-09-10.

[5] "Clickjacking," https://bit.ly/2ZUPSxa, accessed on 2019-09-10.

[6] "Electronic communications privacy act of 1986 (ecpa)," http://bit.ly/33odQRz , accessed on 2019-09-10.

[7] "Internet security threat report," https://symc.ly/2rzm4c5 , accessed on 2019-09-10.

[8] "Kernel Patch Protection, howpublished = https://goo.gl/s4idr7, note = Accessed on 2019-09-10,."

[9] "Khobe 8.0 earthquake for windows desktop security software," https://goo.gl/5UhzpQ, accessed on 2019-09-10.

[10] "Login detection," https://bit.ly/2N0t7Cw, accessed on 2019-09-10.

[11] "Plague in (security) software drivers," https://goo.gl/kmycvb, accessed on 2019-09-10.

[12] "Sleuth kit," https://bit.ly/2sLenAS , accessed on 2019-09-10.

[13] "Ui automation providers," http://bit.ly/37Ib5hi , accessed on 2019-09-10.

[14] "Volatility: An advanced memory forensics framework," https://bit.ly/2s18miV , accessed on 2019-09-10.

[15] "Event tracing for windows," http://bit.ly/2EbzKxM, 2019, accessed on 2019-09-10.

[16] "Hidden treasure: Intrusion detection with etw," http://bit.ly/2VG0DUZ, 2019, accessed on 2019-09-10.

[17] "Lowest common ancestor," https://bit.ly/2udA2lD, 2019, accessed on 2019-09-10.

[18] "Many watering holes, targets in hacks that netted facebook, twitter and apple," http://bit.ly/30otNqs, 2019, accessed on 2019-09-10.

[19] "Plaso: super timeline all the things," http://bit.ly/2WP3MyC, 2019, accessed on 2019-09-10.

[20] "Spear phishing campaign targets ukraine government and military," https://bit.ly/2XaiUGu, 2019, accessed on 2019-09-10.

[21] "Target to pay $18.5m for 2013 data breach that affected 41 million consumers," http://bit.ly/2W77ZQU, 2019, accessed on 2019-09-10.

[22] "Trickbot adds remote application credential-grabbing capabilities to its repertoire," http://bit.ly/2VubtZk, 2019, accessed on 2019-09-10.

[23] A. Developer, "NSAccessibility," https://apple.co/2w2z8Ww, 2019, accessed on 2019-09-10.

[24] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "Saql: A stream-based query system for real-time abnormal system behavior detection," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 639–656.

[25] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling efficient attack investigation from system monitoring data," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 113–126.

[26] Google, "Timesketch: Collaborative forensic timeline analysis," http://bit.ly/2JmCnAT, 2019, accessed on 2019-09-10.

[27] T. U. S. Government, "It accessibility laws and policies," http://bit.ly/2VFOwaD, 2019, accessed on 2019-09-10.

[28] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Network and Distributed Systems Security Symposium (NDSS 18)*, 2018.

[29] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium (NDSS 19)*, 2019.

[30] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. Venkatakrishnan, "Sleuth: Real-time attack scenario reconstruction from cots audit data," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 487–504.

[31] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller, "Dependence-preserving data compaction for scalable forensic analysis," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1723–1740.

[32] K. Jee, G. Portokalidis, V. P. Kemerlis, S. Ghosh, D. I. August, and A. D. Keromytis, "A general approach for efficiently accelerating software-based dynamic data flow tracking on commodity hardware," in *Network and Distributed Systems Security Symposium (NDSS 12)*, 2012.

[33] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 17)*, 2017, pp. 377–390.

[34] S. T. King and P. M. Chen, "Backtracking intrusions," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 223–236, 2003.

[35] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality." in *Network and Distributed Systems Security Symposium (NDSS 05)*, 2005.

[36] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie *et al.*, "Mci: Modeling-based causality inference in audit logging for attack investigation," in *Network and Distributed Systems Security Symposium (NDSS 18)*, 2018.

[37] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *Network and Distributed Systems Security Symposium (NDSS 13)*, 2013.

[38] ——, "Loggc: garbage collecting audit log," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 13)*, 2013, pp. 1005–1016.

[39] B. Li, P. Vadrevu, K. H. Lee, and R. Perdisci, "Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions." in *NDSS*, 2018.

[40] B. Linux, "ATK Package," http://bit.ly/2WJM92Y, 2019, accessed on 2019-09-10.

[41] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security," in *Network and Distributed Systems Security Symposium (NDSS 18)*, 2018.

[42] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC 15)*, 2015.

[43] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha, "Kernel-supported cost-effective audit logging for causality tracking," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 241–254.

[44] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple perspective attack investigation with semantic aware execution partitioning," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1111–1128.

[45] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting." in *Network and Distributed Systems Security Symposium (NDSS 16)*, 2016.

[46] Microsoft, "UI Automation - Windows applications," http://bit.ly/2Q4Vn7v, 2018, accessed on 2019-09-10.

[47] ——, "UI Automation Events Overview," http://bit.ly/2WMJ8yQ, 2018, accessed on 2019-09-10.

[48] Microsoft, "Property identifiers," https://bit.ly/2tCxDQO, 2019, accessed on 2019-09-10.

[49] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*.

[50] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems," in *2006 USENIX Annual Technical Conference (USENIX ATC 06)*, 2006, pp. 43–56.

[51] Panda, "Platform for architecture-neutral dynamic analysis," http://bit.ly/2W5KaJc, 2019, accessed on 2019-09-10.

[52] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "HERCULE: attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the*

32nd Annual Computer Security Applications Conference (ACSAC 16), 2016.

[53] B. Saltaformaggio, Z. Gu, X. Zhang, and D. Xu, “Dscrete: Automatic rendering of forensic information from memory images via application logic reuse,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[54] TinyTask, “Simple + fast + free: Automation for everyone,” http://bit.ly/2vXwrWm, 2019, accessed on 2019-09-10.

[55] Wikipedia contributors, “Cve-2018-8495,” http://bit.ly/30nvxjN, 2019, accessed on 2019-09-10.

[56] ——, “Macro virus,” http://bit.ly/2w3W9Z2, 2019, accessed on 2019-09-10.

[57] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, “High fidelity data reduction for big data security dependency analyses,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 16)*, 2016, pp. 504–516.

APPENDIX

TABLE VI. TESTED WEBSITE LIST

Accommodation and Hotels	www.vrbo.com	www.pegipegi.com	www.ihg.com	www.thetrainline.com
	www.marriott.com	www.airbnb.com.au	www.hotels.com	www.hyatt.com
	www.airbnb.ru	www.trivago.com.tr	www.airbnb.de	www.travelzoo.com
	www.expedia.it	www.airbnb.co.uk	www.choicehotels.com	hotelscan.com
	www.abritel.fr	www.airbnb.fr	www.caesars.com	www.costcotravel.com
	www.oyorooms.com	www.airbnb.com.br	www.trivago.de	secure.accorhotels.com
	magazine.trivago.it	www.wyndhamhotels.com	www.agoda.com	www.fewo-direkt.de
	www.airbnb.ca	www.jalan.net	magazine.trivago.es	www.booking.com
	www.trivago.co.uk	www.homeaway.com	blog.couchsurfing.com	hiltonhonors3.hilton.com
	www.expedia.co.uk	www.airbnb.it	www.hostelworld.com	www.secretescapes.com
	www.trivago.com.br	www.airbnb.es	www.gosur.com	discover.expediapartnercentral.com
Consumer Electronics	sharp.cn	www.made-in-china.com	consumer.huawei.com	www.sony.jp
	www.samsung.com	www.lavamobiles.com	na.panasonic.com	www.boulanger.com
	www.mediamarkt.de	www.micromaxinfo.com	www.apple.com	www.gearbest.com
	www.lg.com	www.mediaexpert.pl	shop.huawei.ru	www.darty.com
	www.pccomponentes.com	www.mediamarkt.es	www.vivo.com.cn	www.vatanbilgisayar.com
	www.onlinetrade.ru	www.uscellular.com	www.oneplus.com	www.mvideo.ru
	www.saturn.de	www.kimovil.com	www.newegg.com	unity.com
	www.gadgetsnow.com	www.shutterfly.com	www8.hp.com	hd.oppo.com
	www.sonymobile.com	www.roku.com	www.currys.co.uk	www.bhphotovideo.com
	www.euro.com.pl	www.mi.com	www.vmall.com	eshop.htc.com
	www.vodafone.de			
E-commerce and Shopping	www.ebay-kleinanzeigen.de	www.wildberries.ru	search.jd.com	hz.58.com
	shopping.yahoo.co.jp	www.olx.ua	www.amazon.es	www.target.com
	shopee.co.id	www.leboncoin.fr	www.ebay.com	articulo.mercadolibre.com.ar
	www.flipkart.com	sale.aliexpress.com	www.amazon.de	www.amazon.co.jp
	www.ebay.co.uk	www.olx.pl	www.ebay.com.au	www.amazon.fr
	err.tmall.com	market.yandex.ru	www.hepsiburada.com	www.sahibinden.com
	listado.mercadolibre.com.mx	www.amazon.co.uk	www.tokopedia.com	www.etsy.com
	allegro.pl	list.tmall.com	www.alibaba.com	login.tmall.com
	kakaku.com	www.amazon.com	slickdeals.net	www.bukalapak.com
	www.ebay.de	produto.mercadolivre.com.br	world.taobao.com	www.amazon.ca
	www.amazon.it	hongkong.craigslist.org	miao.tmall.com	www.dmm.com
	www.amazon.in	www.walmart.com	www.avito.ru	
Finance	www.binance.com	cn.investing.com	www.tradingview.com	www.fidelity.com
	www.investopedia.com	www.schwab.com	www.boursorama.com	iqoption.com
	www.moneycontrol.com	www.alipay.com		
Investing	olymptrade.com	pit.blockchain.com	www.advfn.com	www.ig.com
	new.nasdaq.com	etherscan.io	www.rakuten-sec.co.jp	jfinfo.com
	www.etoro.com	us.etrade.com	www.aastocks.com	finance.eastmoney.com
	yuanchuang.10jqka.com.cn	www.forexfactory.com	www.rico.com.vc	www.kitco.com
	robinhood.com	news.cnyes.com	finviz.com	www.fool.com
	nifty50etf.nseindia.com	www.xm.com	stocktwits.com	www.morningstar.com
	www.finanzen.net	minkabu.jp	nikkei225jp.com	www.tdameritrade.com
	www.ml.com	zerodha.com	about.vanguard.com	www1.oanda.com
	xueqiu.com	www.clear.com.br		
Maps	www.viamichelin.it	www.falk.de	wikiroutes.info	2gis.ru
	fr.mappy.com	www.driveplaza.com	www.onefivenine.com	actualite.lachainemeteo.com
	www.meteofrance.com	economia.uol.com.br	moscow.flamp.ru	mapfan.com
	www.mapquest.com	www.openstreetmap.org	www.targeo.pl	www.langenscheidt.com
	www.city-data.com	www.bustime.ru	www.mapion.co.jp	www.arasikackm.com
	www.marinetraffic.com	www.viamichelin.fr	en.mapy.cz	www.here.com
	www.geoportail.gouv.fr	www.tuttocitta.it	map.goo.ne.jp	www.komoot.de
	www.trendsmap.com	geoguessr.com	www.worldatlas.com	ekitan.com
	www.viamichelin.de	ditu.amap.com		
News and Media	www.naver.com	news.finance.yahoo.co.jp	sohu.com	www.ifeng.com
	www.ukr.net	timesofindia.indiatimes.com	www.rambler.ru	edition.cnn.com
	matome.naver.jp	www.nytimes.com	www.toutiao.com	sina.cn
	elpais.com	www.goo.ne.jp	finance.yahoo.com	www.yidianzixun.com
	www.globo.com	news.yahoo.com	www.iqiyi.com	www.sina.com.cn
	www.yahoo.co.jp	www.livedoor.com	finance.sina.com.cn	www.163.com
	drudgereport.com	map.yahoo.co.jp	www.bbc.com	www.tribunnews.com
	sports.news.naver.com	www.onet.pl	news.google.com	economictimes.indiatimes.com
	section.blog.naver.com	www.ndtv.com	www.qq.com	www.indiatimes.com
	news.yahoo.co.jp	news.mail.ru	search.yahoo.co.jp	www.foxnews.com
	www.wp.pl	www.bild.de	www.t-online.de	headlines.yahoo.co.jp
	www.interia.pl	www.washingtonpost.com	www.msn.com	www.dailymail.co.uk
	us.yahoo.com	www.yahoo.com	vnexpress.net	www.rt.com
	www.uol.com.br	lenta.ru	map.naver.com	www.detik.com
	www.theguardian.com			

Restaurants and Delivery	www.odyssys.net	www.just-eat.co.uk	www.swiggy.com	www.doordash.com
	www.wongnai.com	popslotscasino.com	www.inmoment.com	www.subway.com.hk
	www.seamless.com	www.ubereats.com	www.skipthedishes.com	www.mcdonalds.com
	www.chick-fil-a.com	www.zomato.com	www.opentable.com	dominos.com.mx
	menu.wendys.com	www.lieferando.de	sg.theasianparent.com	www.bk.com
	www.pizzahut.com.hk	zmenu.com	www.chowhound.com	www.olivegarden.com
	postmates.com	www.icomera.com	order.littlecaesars.com	hds-streaming.com
	www.pyszne.pl	www.starbucks.com	www.singleplatform.com	deliveroo.co.uk
	www.grubhub.com	www.mcdonalds.co.jp		
Search Engines	www.google.ca	www.google.co.in	special-offers.online	www.so.com
	www.baidu.com	www.justdial.com	www.startpage.com	www.google.co.th
	www.google.fr	www.google.com.mx	www.ask.com	www.google.com.au
	www.sogou.com	www.google.be	www.google.ru	search.daum.net
	www.google.com.ar	www.google.com.vn	www.bing.com	www.google.de
	www.google.nl	yandex.ru	www.google.com.pe	map.baidu.com
	www.google.ro	www.google.pl	duckduckgo.com	www.google.co.jp
	www.google.com.tr	www.google.com.tw	www.google.com.hk	www.wjx.cn
	m.sm.cn	www.google.es	www.google.com.ua	yandex.com.tr
	search.myway.com	www.google.it	www.google.co.uk	www.google.cl
	www.google.com.br	y2mate.com	whatismyip.li	www.hao123.com
	search.seznam.cz			
Social Networks	ostrovok.ru	www.reddit.com	namu.wiki	incorrect-good-omens.tumblr.com
	programminghumour.tumblr.com	story.snapchat.com	www.pinterest.com.mx	www.pinterest.de
	www.slideshare.net	www.shafa.com	ask.fm	www.pinterest.fr
	bakusai.com	www.pinterest.co.uk	www.znds.com	enterprise.foursquare.com
	ok.ru	www.whatsapp.com	www.weibo.com	vk.com
	www.messenger.com	www.pp.cn	perple.exblog.jp	lihkg.com
	twitter.com	www.facebook.com	www.ptt.cc	www.dangbei.com
	zhuanlan.zhihu.com	www.linkedin.com	ameblo.jp	www.meetup.com
	medium.com	www.instagram.com	www.tagged.com	www.fiverr.com
	www11.eyny.com	www.pinterest.com	www.pinterest.es	zalo.me
	www.dcard.tw			
Sports	sports.yahoo.com	www.yr.no	lequipe.fr	www.americanas.com.br
	www.gazzetta.it	www.kooora.com	www.mundodeportivo.com	www.espncricinfo.com
	www.marca.com	www.goal.com	bbs.hupu.com	as.com
	global.espn.com	www.championat.com	www.mlb.com	www.skysports.com
	www.cricbuzz.com			
TV Movies and Music	www.nicovideo.jp	www.crunchyroll.com	soundcloud.com	www.dailymotion.com
	www.imdb.com	www.google.com	www.youtube.com	www.netflix.com
	www.vesti.ru	vimeo.com		
Others	stackoverflow.com	Xinhuanet.com	Yahoo.co.jp	www.bestbuy.ca
	www.trivago.hk	bbs.tianya.cn	www.twitch.tv	outlook.live.com
	www.papajohnschina.com	airregi.jp	en.wikipedia.org	www.kawauchisyun.com
	templates.office.com	www.gojek.com	www.fishbowl.com	www.okezone.com
	www.360.cn	acomputerblog.blogspot.com	www.csdn.net	www.att.com
	www.microsoft.com	Babytree.com		