Mnemosyne: An Effective and Efficient Postmortem Watering Hole Attack Investigation System

Joey Allen
Georgia Institute of Technology

Zheng Yang
Georgia Institute of Technology

Matthew Landen
Georgia Institute of Technology

Raghav Bhat
Georgia Institute of Technology

Harsh Grover
Georgia Institute of Technology

Andrew Chang
Georgia Institute of Technology

Yang Ji
Palo Alto Networks

Roberto Perdisci
Georgia Institute of Technology
University of Georgia

Wenke Lee
Georgia Institute of Technology

ABSTRACT

Compromising a website that is routinely visited by employees of a targeted organization has become a popular technique for nation-state level adversaries to penetrate an enterprise's network. This technique, dubbed a "watering hole" attack, leverages a compromised website to serve as a stepping stone into the true victims' network. Despite watering hole attacks being one of the main techniques used by attackers to achieve the initial compromise stage of the cyber kill chain, there has been relatively little research related to detecting or investigating complex watering hole attacks. While there is existing work that seeks to detect malicious modifications made to an otherwise benign website, we argue that simply detecting that the website is compromised is only the first stage of the investigation. In this paper, we propose Mnemosyne, a postmortem forensic analysis engine that relies on browser-based attack provenance to accurately reconstruct, investigate, and assess the ramifications of watering hole attacks. Mnemosyne relies on a lightweight browser-modification-free auditing daemon to passively collect causality logs related to the browser's execution. Next, Mnemosyne applies a set of versioning techniques on top of these causality logs to precisely pinpoint when the website was compromised and what modifications were made by the adversary. Following this step, Mnemosyne relies on a novel user-level analysis to assess how the malicious modifications affected the targeted enterprise and seeks to identify exactly which employees fell victim to the attack. Throughout our extensive evaluation, we found that Mnemosyne's forensic analysis engine was able to identify the true victims in all 7 real-world watering hole scenarios, while also reducing the amount of manual analysis required by the forensic analyst by 98.17% on average.

KEYWORDS

attack provenance; watering hole attack; forensic analysis

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3423355

ACM Reference Format:
Joey Allen, Zheng Yang, Matthew Landen, Raghav Bhat, Harsh Grover, Andrew Chang, Yang Ji, Roberto Perdisci, and Wenke Lee. 2020. Mnemosyne: An Effective and Efficient Postmortem Watering Hole Attack Investigation System. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3372297.3423355

1 INTRODUCTION

Sophisticated, targeted attacks against enterprise networks have been growing more frequent recently. Such attacks often unfold through a sequence of steps sometimes referred to as the *cyber kill chain* [65]. To deliver the initial attack that compromises the targeted network, attackers leverage different techniques such as sending spear phishing emails to gain a foothold on the victim's workstation or hoaxing the user to visit a website controlled by the adversary and completing a drive-by-download attack. These tactics have been well studied and enterprises have deployed effective blacklist-based firewall rules (e.g., WAF [10] and Email Defender [4]) and set up periodic security training for its employees. Unfortunately, adversaries have evolved their techniques to infiltrate a targeted enterprise network by compromising whitelisted, third-party entities with which the enterprise normally communicates. For instance, compromising a website that is frequently visited by individuals affiliated with the targeted organization has become a growing trend to achieve the initial intrusion into a targeted organization's network. Such attacks are referred to as "watering hole" attacks. Recently, watering hole attacks have been employed in multiple state-level cybercrimes to conduct digital-espionage in southeast Asia [41], steal proprietary information from large tech firms such as Google and Apple [16, 53], and leak confidential financial information in Poland and Mexico [60, 68].

Despite watering hole attacks being a key method for achieving the initial compromise into an organization, little research has been conducted to study how to detect, analyze, and investigate these attacks effectively. However, having the capability of completing a thorough postmortem analysis is desired by organizations, since it allows them to understand the attacker's intentions, prevent additional damage, and provides a mechanism for building future defenses. In this paper, we propose Mnemosyne, a system that facilitates comprehensive internal forensic investigation on web-based watering hole attacks.

Detecting watering hole attacks and reconstructing their provenance is challenging. First, there has been a significant amount of work dedicated to attack reconstruction, with most solutions relying on whole-system provenance tracking and attack reconstruction [12, 18, 20, 22, 23, 25, 26, 29, 33, 39, 40, 42, 45, 48, 49, 51, 52, 59, 62, 63, 66]. These systems typically collect audit logs that track the information-flow at the system-level and tend to rely on low-level semantics (processes, socket IO, and system calls), which is necessary in order for them to support as many applications as possible. Unfortunately, existing systems' reliance on low-level semantics limit their capability of reconstructing sophisticated watering hole attacks. This is because capturing information at the system level is limited in terms of its capability of understanding fine-grained details related to Javascript (JS) execution within the browser. To overcome this semantic gap, Mnemosyne collects audit logs that capture information in terms of browser-level semantics (e.g. page, script, domain, etc.). While prior browser auditing systems exist [43, 58, 61], they require extensive modifications to the browser itself, making deployment in a real-world scenario difficult. In contrast, Mnemosyne relies on a browser-modification-free, lightweight approach that takes advantage of existing debugging interfaces already provided by off-the-shelf Chromium-based browsers (e.g., Chrome, Opera, Microsoft Edge, Brave, etc.).

The second challenge in developing a system for investigating watering hole attacks is that watering hole attacks are highly targeted and during the early stages of a forensic investigation, it is unclear which visitors are considered the true targets. To address this challenge, we argue that simply detecting a website is compromised is not enough. Instead, for organizations that routinely visit this compromised website, they need to complete a independent and accurate investigation to determine if visiting this site while it was compromised had any adverse effects on their own enterprise networks. However, completing this investigation in an independent manner is not straightforward, since the server-side logs related to how and when the compromised website was modified are external and inaccessible to the targeted organization. To overcome this, Mnemosyne relies on a lightweight auditing approach that passively collects audit logs during a user's browsing sessions. Finally, while Mnemosyne completes the investigation in a postmortem fashion, it is still necessary to complete the investigation in a time-sensitive manner. This is because during the investigation, the decreased system uptime can easily cost millions of dollars [1]. Additionally, an efficient investigation that allows the investigators to quickly identify the scope of the attack can reduce the overall damage created by the attack. To make the investigation as efficient as possible, Mnemosyne applies a set of differential analysis techniques on the audit logs collected to quickly identify which employees at the organization should be considered victims of the attacks and which employees were unaffected by the attack.

In summary, we make the following main contributions:

- **A watering hole attack investigation system.** We propose Mnemosyne[1], a system that is able to accurately reconstruct the provenance and impact of sophisticated watering hole attacks.
- **Browser-modification-free design.** Mnemosyne does not require browser code modifications and can therefore be more easily deployed within users' browsers for collecting detailed web audit logs.
- **Accurate and efficient analysis.** Mnemosyne applies a set of versioning and prioritization methods to efficiently reconstruct and analyse enterprise-level watering hole attacks. Using seven scenarios based on real-world security incidents involving watering hole attacks, Mnemosyne is able to identify the individuals who were victims of the attack in all of these scenarios with efficient runtime.

[1]The source code, datasets, and testbed of Mnemosyne will be made available to the community.

2 MOTIVATING EXAMPLE & CHALLENGES

In this section, we describe an attack scenario modeled off of a real-world watering hole campaign that showcases the challenges that Mnemosyne addresses.

2.1 2017 ASEAN Watering Hole Attack

This case study relates to a real-world, politically-motivated campaign that leveraged a watering hole attack to achieve digital surveillance and espionage on employees and high-powered individuals affiliated with the Association of Southeast Asian Nations (ASEAN), an organization that helps to foster peace between member countries. The attack was carried out in 2017 and was recently attributed to APT32, a nation-state threat actor that is known to carry out cyberattacks against political enemies of the Vietnamese Government [41]. We chose this motivating example because it clearly demonstrates the challenges a forensic analysis will face during postmortem analysis of complex watering hole attacks.

The attack was divided into two stages. The first stage performed reconnaissance by collecting sensitive information related to the user's browser and underlying system to accurately identify if this visitor matched the profile of the targeted victims. After a visitor's profile was developed, it was used to identify the targeted visitors. Finally, the targeted visitors were exploited using social-engineering that pursued victims to grant the attackers access to their Gmail accounts via a malicious OAuth App.

2.2 Challenges

Next, we discuss the challenges that a forensic analyst faces when completing an internal investigation on a sophisticated watering hole attack and discuss the limitations of existing postmortem analysis systems. The challenges described in this section are the challenges faced by the targeted organization, not the organization hosting the compromised website.

External Point-of-Compromise. During a traditional investigation of a sophisticated attack, the initial point-of-compromise occurs at an endpoint system in the enterprise's network. The intrusion is usually achieved through spearphishing emails or traditional exploitation techniques. One advantage this provides is that the audit logs related to this attack will be accessible to the victim organization. Unfortunately, this is not the case for watering hole attacks, since the point-of-compromise is external, beginning at a third-party website that is unlikely to be affiliated with the true victim organization. This creates the challenge that the audit logs

related to the compromised website and the attacker's modifications to the site are only accessible by the website's maintainers, not by the forensic investigator. Due to this limitation, we found that forensic investigators often have to fallback on internet archive sites, such as *archive.org* or passiveTotal [6], to identify how the compromised domain was modified [11, 28, 35, 36]. However, because watering hole attacks are highly targeted, the machines used for snapshotting the web page will not match the intended victim profile. Finally, since the forensic analyst does not know how the web server was modified, the analyst will begin the investigation with minimal information about the initial compromise.

Also, due to the lack of server-side logs, it is challenging to identify the dwell time, which is the time window in which the attacker controlled the compromised website. Identifying the window-of-compromise is necessary to ensure a comprehensive investigation, since any visit to the compromised website within this time window may have led to a successful attack. Without a clear window-of-compromise, the analyst may have to review irrelevant website traffic logs generated prior to the incident, prolonging the investigation.

Semantic Gap. Another limitation is the semantic gap that exists when completing a postmortem analysis on web-based attacks using only system-level logs. Recently, whole-system provenance auditing has been shown to be effective at investigating sophisticated attacks [20, 29, 30, 42, 48, 49]. These systems typically collect audit logs that track the information-flow at the system-level and rely on low-level semantics to causally connect all artifacts and resources involved in the attack. However, using low-level semantics limits reconstruction of watering hole attacks because the semantic gap between system-level and browser based semantics prevents a thorough understanding of JS execution. Prior work that has attempted to address this limitation requires extensive modifications to the browser itself [43, 58, 61], which makes deployment in most enterprises difficult.

Highly Targeted. Watering hole attacks are highly targeted and the granularity of the adversary's targets varies with different attacks. For example, watering hole attacks may only target specific victims at an organization, specific departments of an organization, or a set of organizations. Unfortunately, during the early stages of an investigation, the motive of the attack is unknown and identifying who is targeted by the attack and which individuals fell victim to the attack is challenging. However, identifying the victims of the attack is arguably the most important part of the forensic investigation, since it allows the forensic analyst to determine which user sessions they should prioritize. Also, when the forensic analyst spends time investigating logs related to untargeted users, the investigation is prolonged.

3 MNEMOSYNE

3.1 Overview

Mnemosyne is a forensic analysis engine that completes a postmortem analysis from within the targeted organization with minimal external information. The only information that Mnemosyne requires is the domain name of the compromised website. This design choice was made based on the fact that during the early stages of the investigation, information related to malicious domains used by the adversaries or the modifications made to the compromised domain will be limited and potentially inaccessible to the forensic investigator. Also, the audit logs related to how the website was compromised are external, and in some real-world cases, communication between all entities involved was limited [7].

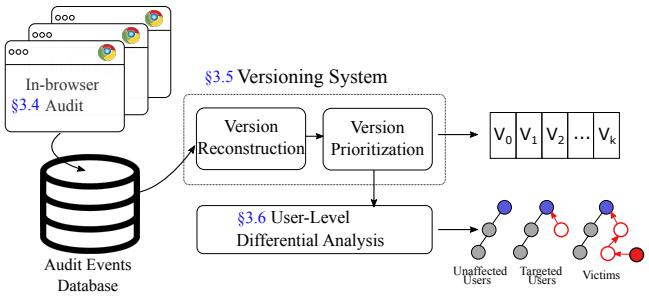

Figure 1: Overview of Mnemosyne's architecture.

An overview of our system is provided in Figure 1, which illustrates its three essential components. The first component is the browser auditor daemon, which is deployed on each endpoint system at an organization to monitor web-browsing activities. The auditor daemon passively collects audit logs throughout the user's browsing sessions without the need to alter the browser. Next, the audit logs from each endpoint are collected and stored on a backend server responsible for maintaining security auditing information.

The next module of Mnemosyne is the versioning system that tracks and analyzes the external website's behavior. The first phase in the analysis is the domain versioning system, which works with the browser-level audit logs to determine when the website was compromised and what modifications the adversaries made to the website. Specifically, the versioning system reconstructs how the compromised website changed over time. Notably, our goal is not to create a single version each time a minor change is observed in the underlying audit logs. Instead, our versioning system helps the forensic analyst quickly identify the window-of-compromise, or the *version* that includes the adversary's modifications to the compromised website that introduced some attack-controlled content.

Next, Mnemosyne provides a version-prioritization approach that prioritizes versions based on their likelihood to be the version that truly represents the window-of-compromise. Developing a prioritization scheme is essential because prior studies have shown that the dwell time can be excruciatingly long, in some cases lasting over 53 months [32]. Meanwhile, benign updates, which will lead to Mnemosyne generating new versions, will also occur, which leads to a challenge in identifying which *domain-version* actually represents the window of compromise and which versions are related to the natural evolution of the website. The last stage of Mnemosyne's analysis is its user-level analysis module, which takes a suspicious domain version and identifies how this domain version behaved differently based on the user that was visiting the site.

3.2 Threat Model and Assumptions

We envision Mnemosyne being deployed in enterprise organizations that have a high risk of being targeted by a sophisticated,

Object Type	Attributes
Frame	**securityOrigin**, sessionId, URL
Iframe	**securityOrigin**, sessionId, URL
Remote Host	**2nd-level domain**, domain
File	**path**, remoteOrigin
Resource	**URI**, type
Script	**hash**, sourceOrigin, URL, sessionId
Session	user-agent, timestamp
HTML Parser	-
User	-

(a) Graph Objects: each object has a unique ID. The bolded attribute represents the object's identifier.

Relationship	Example
Attached	Frame → Frame
CompiledBy	Script → Frame
Created	Script → Frame
Download	Frame → File
Navigated	Frame → Frame
Opened	Frame → Frame
Request	Parser → Resource
Response	Resource → Script
SessionOpened	User → Session
Located	Resource → Host

(b) Relationship between objects.

Table 1: Browser-based provenance graph objects, relationships, and key attributes.

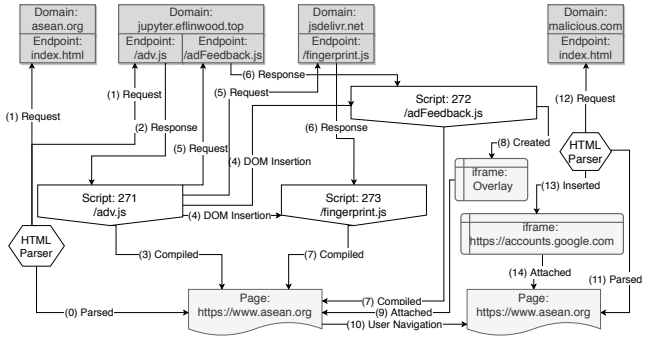

Figure 2: Mnemosyne's browser-based causality log graph of the social-engineering component used by APT-32 to attack targeted visitors.

nation-state-level attacker. Mnemosyne is capable of logging details about users' browsing activities. This means a trade-off between security and privacy must be found. In the envisioned deployment scenarios, it is a reasonable assumption that the executives would be willing to accept a potential reduction to employee privacy to achieve a higher level of security. Furthermore, the audit logs captured by Mnemosyne can be encrypted and securely stored on a file server. A different encryption key can be used for each website and for different time windows. These keys can then be stored in a key escrow, as proposed in previous work [58]. This allows the release of only those keys that are truly needed to enable a forensic investigation.

We also assume that the browser audit logs are stored securely, e.g. using append only log files [15], and thus cannot be tampered with even if the browser is later compromised. Also, we assume that at the time of the watering hole attack, the browser itself is not compromised and the audit logs can be trusted as correct (note that assuming the integrity of the trusted-computing based (TCB) is common in the auditing community [20, 22–27, 29, 30, 33, 34, 39, 40, 42, 43, 45, 47–52, 58, 61, 63, 66]). If the browser is compromised, Mnemosyne can still record correct audit logs related to the attack up until the point when the browser is exploited, thus allowing a forensic analyst to reconstruct the attack setup phase. In our extensive evaluation, we demonstrate that by only recording the "setup" phase of a drive-by download attack campaign Mnemosyne is still capable of identifying the victims of the attack.

3.3 Browser-level Causality Graph

As a first step to enable attack reconstruction, we construct a causality graph based on the browser-level audit logs, which will be used during the postmortem analysis and investigation. The browser objects are defined as nodes in the audit graph, as in Table 1a, and the causal relationships between the objects are defined as edges in the graph, as listed in Table 1b. The graph presents the chain of browser events that occurred and the causal relations they induced. To demonstrate Mnemosyne's capability to reconstruct a web-based attack, we demonstrate how Mnemosyne's logs are able to reconstruct the social-engineering component of the motivating example in Figure 2. The social-engineering attack has three major stages. The first stage fingerprints the user to identify if they match the targeted profile. If so, the second stage blurs the original site's content and injects a malicious overlay into the DOM that redirects the user to a malicious, adversary-controlled site. The final stage of the attack occurs when users navigate to the malicious website that contains a malicious OAuth application. If the user is tricked into granting permission to the malicious OAuth application, it grants the adversaries full access to the user's Gmail accounts.

3.4 Auditor Daemon

To ensure our system can be widely deployed in enterprise settings, we take a different approach than the previous systems that alter the browser (e.g., [43, 58, 61]), and rely on the existing debugging interfaces provided by Chrome that do not require extensive modifications to the browser. Specifically, we rely on Chromium's DevTools interface to extract information about the user's browsing session. Chromium's DevTools Protocol allows tools to instrument, inspect, and profile Chromium, Chrome, and other Blink-based browsers. Mnemosyne's auditor daemon collects the necessary information related to the browser's execution needed to reconstruct the audit logs described in §3.3. A list of the Chrome namespaces used to captured this information is in Table 8 in the Appendix.

3.5 Versioning System

The domain-based versioning system takes in the domain name of the website that is suspected to have been compromised to launch a watering hole attack. The domain versioning system has two major components. First, the *version reconstruction* component reconstructs client-side versions of the compromised domain (§3.5.1). Second, the *version prioritization* component prioritizes the versions in terms of their likelihood of representing the compromised version (§3.5.2). These two components are detailed below.

3.5.1 Version Reconstruction. The first step of the version reconstruction component is to refine the audit graph to only include

TTP ID	Severity	Score	Description & Pattern
T1204.001	Medium	6	**User Execution**: User navigates to an unknown domain. (*p:Page*) − *Navigated* → (*t:Page*) where *p.securityOrigin* = *domain* and *t.securityOrigin* ∉ $D_{profile}$
T1204.002	Very High	10	**User Execution**: User Downloads unknown file. (*p:Page*) − *File-Download* → (*t:File*) where *p.securityOrigin* = *domain* and *t.remoteOrigin* ∈ D_v
T1189	Medium	6	**Initial Access**: Unknown iframe Injection (*s:Script*) − *Inserted* → (*i:Iframe*) − *Attached* → (*p:Page*) where (*s.sourceOrigin* ∈ D_v or *i.securityOrigin* ∈ D_v) and *p.securityOrigin* = *domain*

Table 2: The set of TTPs used in Mnemosyne's weighting system.

pages related to the compromised domain. Specifically, we create the set $P_{domain} = \{p : p.securityOrigin = domain\}$ where *domain* is the compromised domain. Next, for all pages, $p_i \in P_{domain}$, Mnemosyne performs a reachability analysis. The reachability analysis searches the browser causality graph, beginning at p_i, to collect all of the involved objects and network events that occurred when loading the page. This query identifies the domain set, D, which is the set of domains that were communicated with while pages in P_{domain} were loaded into the browser and identifies the earliest and latest timestamps in which network events were made to a domain $d_i \in D$. By extracting the timestamps from the relevant network events, Mnemosyne can reconstruct a chronology profile, which lists the domains in D in descending order by timestamps. Next, the version reconstruction component converts the generated chronology report into versions of the website. The versioning system reconstructs versions based on the domains that were communicated with while the page was loaded into the browser. To construct versions of the website, our system breaks up the chronology report into time windows and then aggregates domains together based on the time the domain first interacted with the website. Specifically, when a set of domains fall within the same time window, they are aggregated into the same *domain-version*, D_v.

Definition 1 (Domain Version). *Given a time window,* $[t_s, t_e]$*, and a webpage,* p_i*, a* ***Domain Version*** $:= \{d \in D : p_i$ *communicated with d for the first time when loading* p_i *in* $[t_s, t_e]\}$

Manually inspecting the domain sets to determine the boundaries of new versions is time-consuming and leads to analysis fatigue [23]. For this reason, we automated this process. We first rely on a profiling phase that identifies the profile domain version, $D_{profile}$, which represents the set of benign domains that are responsible for commonly serving content to visitors of the compromised website. We define the time window required to learn $D_{profile}$ as the *profiling phase*, which has a duration of ω days. We provide a detailed discussion of how to appropriately calculate ω in §8.1. After $D_{profile}$ has been learned, Mnemosyne begins creating new domain versions on the date that a new domain was observed. Additionally, when multiple domains appear in the same day, Mnemosyne will aggregate these domains into the same domain-version.

User-Level Property	Description
Version ID	A unique identifier for this version.
Parent Version ID	The parent version's ID.
Page Set	The set of page IDs assigned to this version.
Δ-Set	The set of objects that were responsible for generating this version.
Size	The number of pages assigned to this version.
User Set	The set of users for this version.

Table 3: The metadata properties of a user-level version.

3.5.2 Version Prioritization. To make Mnemosyne more efficient in locating the window of compromise, we prioritize the domain versions in the order of their likelihood to be the version that truly represents the window-of-compromise. This prioritization analyzes each domain version independently to identify suspicious behavior causally dependent on this domain version. We quantify the suspiciousness of these behaviors using a weighting system. Based on the behaviors found, an overall *suspiciousness* score is defined for the domain version. The domain versions are then placed in a priority queue based on their suspiciousness score. This prioritization focuses the analysis on the most suspicious versions, increasing the investigator's efficiency.

Weighting System. Mnemosyne's weighting system is TTP-based, analogous to existing state-of-the-art, whole-system auditing approaches (e.g., Holmes [52] and Rapsheet [24]) in the sense that it relies on matching browser-based audit logs to existing attack patterns in the MITRE ATT&CK Framework [5]. The set of TTPs Mnemosyne relies on to detect suspicious domain versions and the patterns required to match these TTPs to the browser-level audit logs is defined in Table 2. For each domain version D_v, Mnemosyne calculates a suspiciousness score. First, Mnemosyne conducts a reachability analysis, starting from the set of domains in D_v, to identify the set of pages impacted by this domain version, which we call $P_{affected}$. For each page in $P_{affected}$, we search for audit events that match a TTP pattern. The score of the domain version is the sum of the severity scores of the matched TTPs defined in the *Score* column of Table 2.

3.6 User-Level Analysis

The final stage in Mnemosyne's analysis is the user-level analysis. The purpose of the user-level analysis is to identify how a domain-version behaved differently based on the user that was visiting the compromised website, with the ultimate goal of minimizing the effort required by the forensic analyst to determine which users were unaffected, targeted, or victims of the attack. For each domain version, D_v, pulled from the priority queue, and the set of pages, $P_{affected}$, associated with it, the user-level analysis clusters pages in $P_{affected}$ that had similar behaviors while they were loaded into the browser. This clustering minimizes the number of pages the FA needs to analyze by aggregating pages with similar behaviors. The FA can then analyze clusters as one and assess and make decisions about entire clusters of pages instead of only a single page, reducing the amount of effort and time required to complete the investigation.

The clustering approach has two stages. The first stage extracts a features set from each page in $P_{affected}$. In the second stage, a differential analysis is completed over the feature sets extracted.

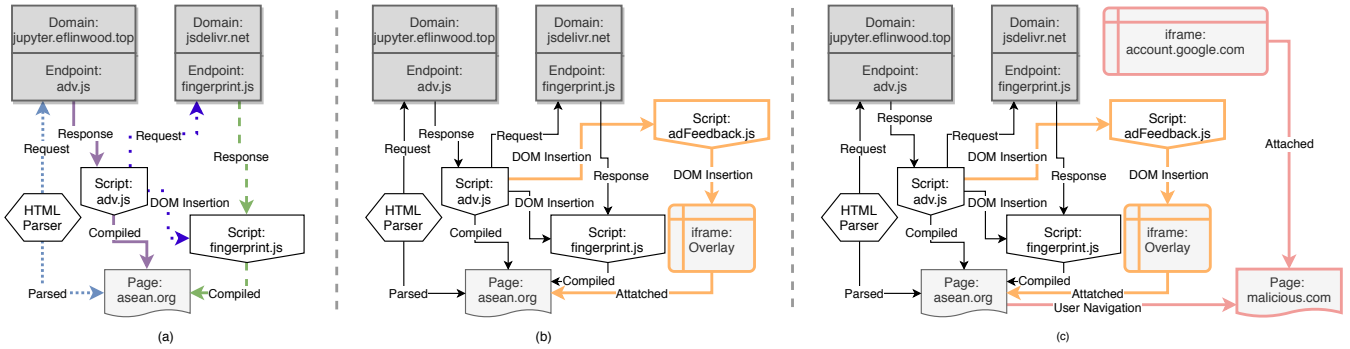

Figure 3: Three subgraphs related to visiting *hxxps://www.asean.org* during the window-of-compromise. (a) represents a visit by a non-targeted user, (b) represents a visit by a targeted user, and (c) represents a visit by a victim of the attack.

The differential analysis phase generates clusters of pages. We call these clusters *user-level versions*; the term cluster and user-level version are used interchangeably.

Definition 2 (User-Level Version). *A **User-Level Version** := $\{p_i \in P_{affected}, metadata(p_i) : all\ p_i\ share\ the\ same\ feature\ set\}$.*

For each user-level version, a set of metadata properties, defined in Table 3, is tracked. We will provide additional information about each property during the remainder of this section. The final step of the differential analysis is to insert the user level versions into a versioning tree, where nodes are user level versions and edges represent dependencies between the versions. The benefit of this version tree is that it orders the versions and allows the forensic analyst to quickly determine what modifications were made to create the new version and the ancestry of each version.

3.6.1 Feature Extraction Phase. For each page p_i in $P_{affected}$, we extract a set of features. Specifically, Mnemosyne collects all paths from the domains in D_v to the page p_i by querying the audit graph. The result of this query is the set of all paths, $PATHS_{d \rightarrow p_i}$, where $d \in D_v$. For example, in Figure 3.a we see the results of this query on the domain version, D_v = { *jupyter.elfinwood.top, jsdeliver.net* }. It returns four paths (cyan, purple, green, and blue). The set of paths returned from the causality query are combined to create a subgraph, S_{p_i}. This process is repeated for all pages in $P_{affected}$. Figure 3 shows the subgraphs for three example pages. (a) represents a page where the user was unaffected by the attack, (b) represents a page where the user was targeted, which is highlighted in orange, and (c) represents a page where the user was a victim of the attack, which is highlighted in pink. The next step converts the subgraph, S_p to a feature set, $\hat{S}_p$. For each node in S_p, we extract its identifier. The identifier for each node is the bolded attribute in Table 1a. For the edges, we create a three tuple containing the relationship type, and the identifier of the start and end nodes. The feature set consists of the nodes' identifiers and the relationship tuples for the page p_i. This process is completed for all the related subgraphs. After creating $\hat{S}_{p_i}$ for all pages, the next task is to assign each page to its initial user-level version. A page is assigned to its initial user-level version based on its feature sets. Specifically, pages are assigned to a user-level version, U_v, if $\hat{S}_{p_i} == U_v.\Delta$. If no match is found, a new user-level version will be created, this page will be assigned to it, and $U_v.\Delta$ will be set to $\hat{S}_{p_i}$.

3.6.2 Differential Analysis. The final phase in Mnemosyne's analysis is differential analysis. Given the initial user-level version set `UserVersions`, the differential analysis creates a version graph, where each node represents a unique user-level version and the edges represent ancestral relationships between the versions in the graph. For our analysis, an ancestral relationship implies the resources in the parent's Δ-*Set* were also observed by pages in the child's *pageSet*. The advantage of presenting the versions embedded into a version graph is that it allows the FA to assess the modifications and differences made between the child and parent versions, and quickly determine which users observed which behaviors.

The differential analysis is initiated by selecting the root version from the set `UserVersions`, shown on line 5 of Algorithm 1. The selection of the root version is based on size of the user level version's *pageSet*. The version with the largest page set is selected to be the `current` version. The `current` version will then be inserted into the `VersionGraph`. Next, the algorithm iterates over the remaining user-level versions. For each user-level version, $u_v \in$ `UserVersions`, the algorithm will determine if `current` is a parent version of u_v. `current` is considered a parent when the intersection of $u_v.\Delta$-*Set* and *current*.Δ-*Set* is non-empty, as shown on line 10. When a parent version is found, the differential analysis updates the user version's Δ-*Set*. Specifically, a `diff` operation is performed on the user version's delta set, where $u_v.\Delta := u_v.\Delta - current.\Delta$ This operation prevents duplicating the same behaviors, which would increase the FA's workload and prolong the analysis. After updating the Δ-set of u_v, we compare u_v's Δ-set to the remaining user versions' Δ-set. If they are equivalent, we run a `merge` operation and merge the two user-level versions. This step maximizes the cluster sizes, minimizing the feature sets that the FA has to manually analyze. We repeat this process, assigning the version with the largest, remaining page set to *current*, until every user-level version has been inserted into `VersionGraph`. The output of this differential analysis is the `VersionGraph` for the user-level versions, which the forensic analyst can use to quickly assess the different behaviors exhibited by the domain version they are evaluating.

Algorithm 1: Differential Analysis

```
Input: UserVersions The set of initial user-level versions.
Result: VersionGraph: The resulting Versioning Graph.
begin
    VersionGraph ← list();
    while |UserVersions| > 0 do
        // Initialize current based on pageCount.
        current = max(UserVersions.pageCount)
        // Insert current version into VersionGraph.
        VersionGraph.insert(current);
        foreach u_v in UserVersions do
            // Determine if current is a parent of u_v.
            isParentVersion = u_v.deltaSet ∩ current.deltaSet
            if |isParentVersion| > 0 then
                // Add current as a parent to version u_v.
                u_v.parents.append(current.versionId);
                // Complete diff operation on version u_v.
                u_v.deltaSet = u_v.deltaSet - current.deltaSet;
                foreach m_v in UserVersions do
                    if m_v == u_v ∨ m_v == current then continue ;
                    else if u_v.deltaSet == m_v.deltaSet then
                        // Complete Merge on u_v and m_v.
                        u_v.pageSet.append(m_v.pageSet);
                        u_v.userSet.append(m_v.userSet);
                        UserVersions.remove(m_v);
                    end
                end
            end
        end
        // Remove current version from UserVersions.
        UserVersions.remove(current);
    end
    return VersionGraph
end
```

4 EVALUATION

Our evaluation addresses the following research questions:

- How effective is Mnemosyne at reducing the analysis scope of the forensic investigation?
- How does the benign evolution of websites affect Mnemosyne analysis?
- What is the runtime performance overhead of Mnemosyne's auditing daemon and analysis?
- What are the data storage requirements for Mnemosyne?

4.1 Data Collection

The highly-targeted nature of watering hole attacks makes them extremely difficult to detect in the wild, which results in difficulty of collecting data on them. To overcome this, we developed a scalable testbed that is capable of simulating sophisticated watering hole attacks on a large organization. This testbed has the capability to make arbitrary modifications to an otherwise benign website and simulate visits to compromised websites. To simulate a compromised website, we relied on Chromium's DevTool's `Fetch` namespace, which can intercept and modify network requests made by the browser. Our testbed supports directly modifying HTML pages and scripts on-the-fly to support various modification techniques. This allows the testbed to simulate malicious modifications being made to the website. To simulate visits, we developed a driver based on puppeteer [8]. During a visit, the driver navigates to up to 15 webpages on the site. However, only visiting webpages limits execution coverage because modern webpages are highly-dynamic and event-driven. To address this, our driver simulates JS-based events while visiting the page. Also, the driver automatically emulates different browser/OS combinations, including mobile operating systems (e.g., screen size and other system properties are adjusted according to the emulated system). Finally, we designed our testbed to be scalable by making each crawler a container-based application, which allowed us to execute multiple crawlers in parallel. Each container contains a headless Chromium browser and the driver that simulates a visit to the compromised domain.

4.2 Datasets

Leveraging the watering hole testbed discussed in §4.1, we collected datasets to develop 7 attack scenarios discussed in §4.2.1 and two benign datasets discussed in §4.2.2.

4.2.1 Attack Scenarios. To extensively evaluate Mnemosyne, we developed 7 attack scenarios inspired by watering hole attacks that have been reported in the wild. A detailed description of the attack scenarios is provided in Table 4, including the website that we simulated being compromised and a reference to the real-world attack that inspired this scenario. For each attack scenario, we collected data for at least two weeks, and each scenario had three phases. During the first phase, the website was benign, and no malicious modifications were made. The purpose of this phase was to collect the auditing information necessary to model the benign behavior of the website. The second phase simulated a reconnaissance phase, where the website was compromised. At this point, the simulated attacks had not targeted any users. The final stage was the targeting phase, where the attack actively sent the malicious payload to victims. We provide statistics related to the size of the attack graph for each scenario and important crawling statistics in Table 5. On average, we simulated 1,844 visits during the attack scenarios. Also, we found that, on average, 1,941 distinct URLs related to the compromised domain were visited. Finally, the average graph size for each attack scenario was 6.2M and 11.0M nodes and edges respectively.

4.2.2 Benign Datasets. The benign datasets contain simulated visits to a large set of benign websites. We collected two datasets, which we will call "Categories" and "Alexa". The details of each dataset is described below.

Categories. The categories dataset was developed by crawling 900 websites from February 6th, 2020 to August 18, 2020. Each website crawled had an associated category tag, where the category tag represents the website type (e.g., News, Sports, etc.). To categorize the websites, we leveraged DMOZ; the most comprehensive, human-edited directory of the Web [3]. The 900 websites were randomly selected. The number of webpages in each category is provided in Table 9 in the Appendix. After removing websites that returned an error, we had a dataset of 830 valid websites that included 278,177 unique pages related to the websites.

Alexa. The Alexa dataset was developed by crawling the Alexa 1k from July 13th, 2020 to August 18, 2020. In total, we collected 120,245 unique pages related to these websites.

Attack Scenario	Website & Description	Reference
Malicious OAuth Access	*www.cfr.org* — The adversary injected a malicious script into the homepage. If users were targeted, the original content of the page was blurred, and a malicious overlay was injected into the DOM. If the client interacted with the overlay, it redirected them to an attacker-controlled website hosting a malicious OAuth app that requested sensitive email permissions.	[41]
Clickjacking	*www.acumen.org* — The attack embedded a malicious iframe onto the page, which redirected users to a malicious website, hosting a malicious OAuth app. The app requested sensitive Gmail permissions.	[54]
Malicious Software Update	*www.energy.gov* — The adversary injected a malicious flash update onto the webpage. Victims of the attack were tricked into downloading a trojanized version of Adobe Flash.	[38]
Credential Harvesting	*www.cipe.org* — The adversary manipulated the original webpage's DOM to mimic a Google login page. Victims of the attack would be tricked into leaking sensitive credentials.	[11]
Keylogging	*www.xero.com* — The adversary injected a keylogger into the webpage, which logged all keystrokes by targeted clients.	[55]
Tabnabbing	*www.thebanker.com* — The adversary used a tabnabbing attack to distract the user. Next, the attackers injected an iframe, which mimicked the institution's login page. The attack victims leaked their sensitive email credentials.	[21]
Driveby	*www.zingnews.vn* — The adversary injected a malicious script into the homepage. If the users matched the client profile, the malicious script injected a 1x1 iframe into the DOM, which navigated to a malicious website that exploited CVE-2020-6405 to complete a drive-by download attack.	[31]

Table 4: Description of each attack scenario and the corresponding case in the wild.

Attack Scenario	Nodes/Edges	Visit Sessions	Pages Visited	Distinct URLs	Script Instances	Network Events
Malicious OAuth Access	9.20M / 16.1M	5.37K	57.8K	4.96K	8.80M	3.10M
Clickjacking	4.40M / 5.90M	950	7.66K	267	3.99M	658K
Malicious Software Update	628K / 1.30M	1.00K	8.69K	640	547K	374K
Credential Harvesting	770K / 1.60M	927	8.40K	364	710K	449K
Keylogging	20.9M / 33.8M	1.91K	100K	5.33K	18.8M	6.20M
Tabnabbing	6.70M / 15.0M	2.18K	83.2K	741	6.20M	4.30M
Driveby	952K / 3.10M	580	6.72K	1.29K	742K	926K
Average	6.20M / 11.0M	1.84K	39.0K	1.94K	5.7M	2.31M

Table 5: Graph statistics for each attack scenario.

4.2.3 Data and Evaluation Limitations. There are some potential limitations to pay attention to when relying on simulated attack scenarios to complete an evaluation. First, when visiting each site in the attack scenario, the navigation through different pages on this website was randomized. In practice, website visitors will typically follow specific and routine visiting patterns to complete specific tasks. However, Mnemosyne's analysis does not rely on the visiting pattern of the users, so this is not expected to represent a significant issue in practice. Next, our testbed does not support automatically logging into a webpage. Since portions of a website may require authentication to view, some portions of a website may be unavailable to our testbed. While this does limit the visibility of the website in our experiments, it will not be a significant issue in practice. This is because, in a real-world deployment, Mnemosyne would have visibility to these portions of the website once the user logged into the site, since Mnemosyne will record audit logs as the user interacts with webpages through the browser. Finally, one limitation of our testbed is content tailored to a specific user for benign use-cases. While our testbed can simulate different users, this simulation mainly alters the User-Agent string when visiting the website. Unfortunately, for websites that distribute content based on profiling the user or requiring the user to log in, our current implementation of emulating different users will most likely lead to the websites not serving "user-specific" content in a meaningful way. However, we believe benign use-cases of user-specific content will not have a large affect on Mnemosyne's analysis because, while websites routinely serve user-specific content, this content will be served off the same set of domains. Since Mnemosyne would identify these domains during its profiling phase, these user-specific modifications would be filtered out of the analysis scope.

4.3 Attack Scenario Investigation

4.3.1 Forensic Analysis Scope Reduction. To measure the efficiency gains that Mnemosyne provides, we completed an empirical evaluation to quantify how much of the analysis space is reduced when using Mnemosyne to complete the investigation.

Defining the Analysis Space. We define the analysis space as the set of domains and scripts related to each attack scenario in Table 4. We want to point out that the number of scripts reported for each attack scenario is the number of unique script URLs, not to be confused with the number of script instances. The choice to focus on domains and scripts is based on a preliminary study we conducted with 5 security-trained professionals. The purpose of this study was to assess how different professionals approach forensic investigation tasks. To this end, we assigned an investigation task to each participant, provided them access to the browser logs, and asked them to determine the window-of-compromise and the attack's victims. Each participant was provided with access to a graph database that contained the attack scenario logs and a graphical interface[2] for interacting and making queries to the database to enable the investigation. After each participant completed the task, we conducted an exit interview to discuss what strategies the participants adopted to perform the investigation. We found that most participants used a two-phased approach. First, they filtered out well-known domains. Then, for the remaining domains, they analyzed the scripts served by those domains. Because of this approach, we consider the number of domains and scripts involved in the attack scenario to play a larger role in the analysis time compared to other types of resources (e.g., images, CSS files, etc.).

[2]https://neo4j.com/developer/neo4j-browser

	Raw		manual++			Mnemosyne		
Attack Scenario	**# of Domains**	**# of Scripts**	**# of Domains**	**# of Scripts†**	**# of Scripts‡**	**# of Domains**	**# of Scripts**	**# of Versions**
Malicious OAuth Access	59	6,243	45 (-23.73%)	5,814 (-6.87%)	477 (-92.36%)	4 (-93.22%)	6 (-99.90%)	5
Clickjacking	18	523	12 (-33.33%)	507 (-3.06%)	116 (-77.82%)	2 (-88.89%)	5 (-99.04%)	3
Malicious Software Update	23	1,102	20 (-13.04%)	1,086 (-1.45%)	318 (-71.14%)	1 (-95.65%)	63 (-94.28%)	2
Credential Harvesting	20	534	15 (-25.00%)	521 (-2.43%)	112 (-79.03%)	1 (-95.00%)	85 (-84.08%)	2
Keylogging	11	1,761	11 (0.00%)	1,761 (0.00%)	37 (97.90%)	2 (-81.82%)	6 (-99.66%)	3
Tabnabbing	64	27,477	48 (-25.00%)	27,259 (-0.79%)	26,262 (-4.42%)	2 (-96.88%)	13 (-99.95%)	3
Driveby	571	3,230	567 (-0.70%)	3,055 (-5.42%)	350 (-89.16%)	2 (-99.65%)	2 (-99.94%)	3
Average	109	5,838	103 (-6.27%)	5,715 (-2.21%)	4,554 (-22.01%)	2 (-98.17%)	26 (-99.56%)	3

Table 6: A detailed performance comparison between manual++ and Mnemosyne in terms of number of domains and scripts a forensic analyst needs to investigate. Scripts† are all scripts related to the corresponding domain. Scripts‡ are the set of scripts that have behaviors attributed to them (e.g., network requests, DOM insertions, etc.). Mnemosyne can reduce the analysis scope significantly, for example, (-99.56%) means reduction based on the raw data.

Developing a Baseline. Following the practical strategies observed during this study, we developed a baseline system to compare against Mnemosyne, which we call manual++. manual++ attempts to generalize the approaches used by the different participants to perform a forensic investigation based on browser logs. Specifically, manual++ first collects all domains that communicated with the compromised website. Next, it filters out any of these domains that are listed on the Alexa 10k, since they are highly likely to be benign. Next, it further reduces the number of domains by filtering out domains that only served static content (e.g., images, fonts, CSS files, etc.).

Measuring Analysis Reduction. To measure the analysis reduction Mnemosyne provides, we compare it to manual++. We focus on the number of domains and scripts that would require manual inspection when using Mnemosyne, compared to manual++. An extensive reporting of the results is provided in Table 6. The number of domains and unique script URLs found in each attack scenario is reported in the *raw* column. We see that Mnemosyne was able to filter out, on average, 98.17% of the domains while manual++ was only capable of filtering out 6.27% of the domains. It's even less for the case of *Driveby* because the website employs ads that generate random domain names. Next, we inspected the number of scripts filtered out of the analysis space by Mnemosyne and manual++. Our experiments show that Mnemosyne was able to filter out 99.56% of the scripts from the analysis space, which shows a significant reduction in the number of scripts required for manual analysis by the investigator. To measure the number of scripts filtered by manual++, we provide two results in columns *Scripts*† and *Scripts*‡. *Scripts*† provides the number of scripts remaining after applying manual++'s filtering. We see that on average, only 2.21% of scripts were filtered by manual++. Also, we provide a second set of results in column *Scripts*‡. The results presented in column *Scripts*‡ were calculated by adding an additional filtering stage, which was more aggressive and filtered out scripts that did not have behaviors causally attributed to them (e.g., they made no network requests or DOM insertions). The results show that applying this additional filtering stage can reduce the number of scripts by 22.01% on average, and in some cases, such as the Keylogging scenario, this additional filtering stage performs well. However, we also see that in the Tabnabbing attack scenario, it performs extremely poorly, and was only able to filter out 4.42% of the scripts in the analysis space, while Mnemosyne was able to filter out 99.95% of scripts. We also found that the naive approach used by manual++ to filter out domains based on the Alexa 10K led to a false-negative in the *Malicious Software Update* attack scenario, because the adversaries served the malware from a Git repository on *hxxps://www.github.com*. On the other hand, Mnemosyne correctly identifies this attack component. These results show that Mnemosyne can significantly reduce the scope of the analysis space that requires manual analysis for the forensic investigation.

4.3.2 Attack Scenario Domain Versions. Next, we investigated the number of domain versions generated for each attack scenario. For each scenario, we set $\omega = 1$ (i.e., we used one day for the profiling phase). We found that a low number of versions were generated for each attack scenario, as reported in the last column of Table 6. One outlier was the Malicious OAuth Access Scenario, with 4 versions after removing the core domain-version. We further investigated and found that there were 3 new benign versions generated. These versions were generated shortly after the profiling phase, which means this phase was too short for this website. This is reasonable since this scenario compromised *hxxp://www.cfr.org*, which was a larger website and had 4,957 distinct URLs visited during the scenario. Also, the version prioritization prioritized all 3 of these benign versions lower than the malicious version.

4.3.3 Version Prioritization. In all attack scenarios, our system accurately prioritized the compromised version over the benign versions, with the one exception of the Keylogging Attack Scenario. This shows Mnemosyne's version prioritization approach was effective, and we investigated each attack scenario to determine exactly why the prioritization was effective. The Malicious OAuth Access attack lured the user into navigating to *jupyter.elfinwood.top*[3], which was captured as a cross-origin navigation in the causality graph. As this cross-origin navigation was attributed to a script served by *jupyter.elfinwood.top*, it flagged TTP *T1204.001*, which incremented its suspiciousness score. Finally, for the Clickjacking and Tabnabbing scenarios, Mnemosyne identified TTPs *T1189* & *T1204.001*

[3] *jupyter.elfinwood.top* was the malicious domain used throughout the attack scenarios.

Attack Scenarios	User-level Versions					
	Unaff.	Tar.	Vic.	Unaff.-Tar.	Tar.-Vic.	Unaff.-Vic.
Malicious OAuth Access	✓	✓	✓			
Clickjacking	✓	✓	✓			
Malicious Software Update			✓	✓		
Credential Harvesting	✓	✓	✓			
Keylogging	✓	✓	✓			
Tabnabbing	✓	✓	✓			
Driveby	✓	✓	✓			

Table 7: A report of the user-level version types generated during each attack scenarios.

being causally dependent on the attack scenarios' respective malicious domain version. For the Malicious Software Update attack, it successfully detected TTP *T1204.002*. Finally, for the Driveby scenario, we detected TTP *T1189* when the iframe was injected into the DOM. Mnemosyne was not effective in prioritizing the Keylogging attack scenario since the attack did not insert an iframe nor trick users to do anything, but rather sent network requests in the background. We believe this is acceptable, as version prioritization does not aim to determine the compromised version directly but aims to prioritizes domain-versions relying on identifying typical suspicious events that are related to social-engineering attacks.

4.3.4 User-Level Versions. To measure the effectiveness of the User Level Analysis, we determine how effective it is at developing "uniform" clusters. We define a cluster as uniform if the pages mapped to it only represent unaffected users, targeted users, or victims. We define six different types of user-level versions; unaffected users (Unaff), targeted (Tar), and victim (Vic) versions only include a single user-type. Next, the groups unaffected-targeted (Unaff-Tar), targeted-victim (Tar-Vic), and unaffected-victim (Unaff-Vic) are *mixed* user-level versions. Mixed groups are generated by the user-level analysis when there is not enough context in the underlying audit logs to accurately distinguish between the two user types. The results are shown in Table 7, where each column represents a different category of user-level versions. We see that in 6/ 7 attack scenarios, Mnemosyne generates ideal "uniform" clusters. For the Malicious Software Update attack scenario, Mnemosyne created two user-level version types, a victim user-level version and a mixed unaffected-targeted version. To understand this, we analyzed the attack scenario in more depth and found that the attack relies on inserting an `overlay` tag into the page to lure the user into installing the malicious file. Since Mnemosyne relies on an instrumentation-free approach for auditing the browser, it has less visibility in terms of DOM modifications compared to prior work [43] and unfortunately could not attribute the insertion of the `overlay` to a specific script. However, since file download events can be detected, Mnemosyne is able to narrow down the analysis space to identify all the users that were victims of the attack.

4.4 Benign Version Analysis

Next, we completed a study to evaluate the number of versions reconstructed over an extended time period. To achieve this, we evaluated Mnemosyne's domain-version reconstruction using the benign datasets with $\omega = 1$. The average number of versions reconstructed per category is shown as the solid triangle in Figure 4. The light yellow box shows the five-point-summary for the Alexa and Categories dataset. We found that news websites have the highest average with 4.33 versions generated during the crawling period, while gaming sites only had 1.52 versions. The average number of versions generated in one month is 2.15 for the Alexa Top 1k. An extreme outlier, *hxxp://auctiongr.com*, in the shopping category generated 22 versions. This website was a shopping site back in February 2020 and only generated one version till March. However, the domain was registered to a porn and malvertising site in June 2020, and started adding malicious domains frequently. There are 7 websites considered outliers, for the Alexa category, which have more than 6 versions generated in one month. Among the 7 websites are 6 News websites that are listed in Table 11 in the Appendix. We believe this is a reasonable outcome, since websites that fall into the News category need to frequently add new content to stay up-to-date on current events. The other one is a computer-themed forum-like website which updates its content frequently as well. This experimental evaluation shows that Mnemosyne's domain-versioning system can be effective for long periods of time.

4.4.1 False-Positive Analysis. Benign updates made after the profiling phase will generate a new domain version. We completed an experimental evaluation to assess how often benign updates would be flagged as suspicious. To complete this experiment, we used the benign versions discussed in §4.4. We inspected 3,663 benign versions generated across 1,830 websites. We found that 14.12% of the benign versions were flagged as suspicious. We further investigated the flagged benign versions and found that 2.38% of the versions were flagged due to cross-origin navigation to an unknown domain, while 11.74% of the flagged domains were related to anomalous iframes being injected into the DOM. However, in almost all cases the iframes injected were ad-related. We found that by checking the iframe's `src` property, against a white list of 15 domains (listed in Table 14), it allowed us to filter out 9.15% of the iframes related to ads. Finally, after applying the white list filtering approach we find that only 4.97% of the benign versions were flagged as suspicious. Since the analyst will only need to investigate benign versions related to the compromised version, this shows that benign updates will not significantly increase the time of the investigation.

4.5 Runtime Performance

To evaluate the runtime performance of the auditing daemon, we measured the *page load* time for the top 1,000 most popular websites according to Alexa.com, using out-of-the-box Chromium version 80.0.3987.163. The page load metric is important because previous studies have shown that a slow page load time can lead to frustrated users and drive websites' revenue lower [19]. For each site, we conducted 10 trials, both with and without our auditor attached. Prior to measuring the page load time, we loaded the homepage of each site into the browser so that it would heat up the browser's cache. The purpose of this was to minimize the influence that potential network latency variations would have on the experiment. It is important to notice that Mnemosyne logs web requests regardless if the object was fetched from the cache or the actual server, so the time spent within Mnemosyne's logging functions will be the

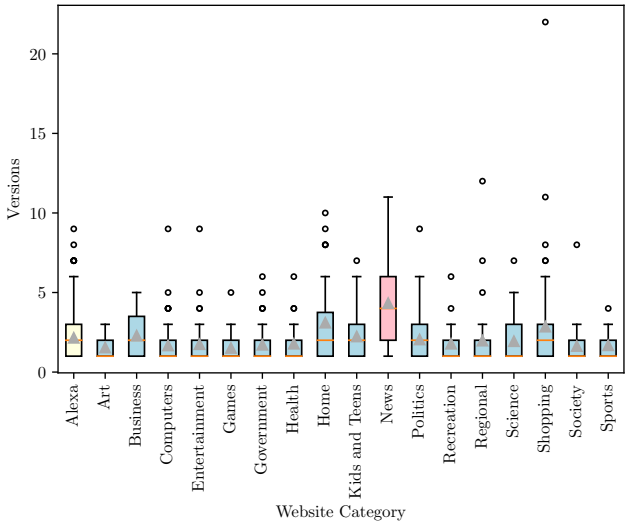

Figure 4: Five-point-summary of domain versions generated for each website for the Categories and Alexa datasets, with whiskers being 1.5x IQR.

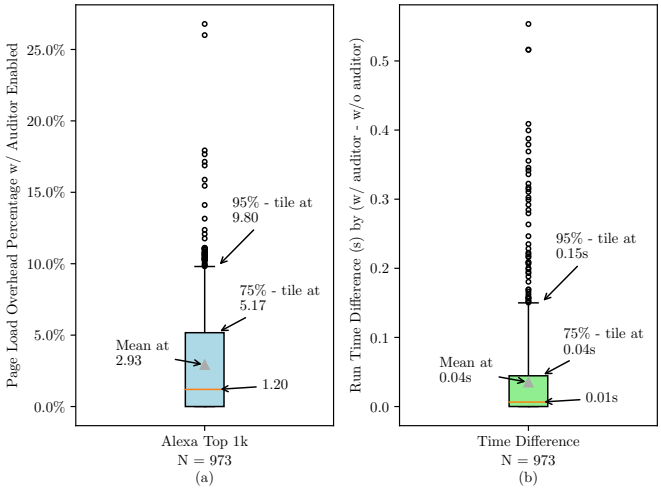

Figure 5: The runtime performance overhead induced on the page load by Mnemosyne for the Alexa 1k. (a) presents the runtime overhead increase for the page load. (b) provides the absolute time induced by Mnemosyne. Whiskers are set to (0%, 95%).

same whether an object is retrieved from the cache or directly from the network.

The experimental results are presented as a five-point-summary in Figure 5, with (a) presenting the overhead percentages compared to loading a page while Mnemosyne is off, and (b) presenting the absolute time induced by Mnemosyne on the page load. We found that Mnemosyne's auditor daemon had a low performance overhead of only 2.93% on average and a 95th-percentile overhead of 9.80%, which is similar to the overhead introduced by previous work [43]. Additionally, (b) shows that on average Mnemosyne increases the load time by only 0.04s. However, we found two outliers, *hxxps://www.tripadvisor.com* and *hxxps://www.atlassian.com*, which had page load overheads that were slightly over 25%. We spent a significant amount of time assessing why these two cases were outliers. This included toggling the DevTools namespace to identify exactly which namespace(s) were causing the performance overhead. We found that the `Network` and `Debugging` DevTools namespaces appear to be contributing the most to the overhead induced. Unfortunately, a more fine-grained approach to identify exactly which other DevTools hooks were contributing to the overhead and by how much would require instrumenting Chromium itself. Since this outlier overhead was observed only on 2 out of 1,000 websites, we leave this detailed analysis to future work.

It is important to notice that Mnemosyne leverages only a small set of DevTools hooks within a small set of namespaces, namely `Network`, `Page`, `Debugger`, and `Target`. Therefore, runtime performance could be further optimized by developing a customized `Forensics` DevTools namespace, which would only activate the hooks that are necessary for the logging, while avoiding the overhead introduced by calls to other unused hooks that occur when other DevTools namespaces are present. In summary, our performance evaluation shows that Mnemosyne has a reasonable overhead, especially for a prototype, and could be deployed in real-world scenarios without significantly affecting the user's browser experience.

Next, we measured the performance of Mnemosyne's automated log analysis process (see §3.5 and §3.6) on a standard laptop with Intel I7-8700B CPU running at 3.2 GHz and 32GB of physical memory. On average, the log analysis process takes less than 5 minutes for a graph of 6.2M nodes and 11.0M edges. Additionally, a breakdown of the runtime performance for every attack scenario is provided in Table 12. This shows the runtime performance for analyzing each attack scenario is efficient.

4.6 Storage Overhead

To measure the disk space overhead, we ran Mnemosyne's auditor for a 50-minute browsing session and visited 10 heavily dynamic and popular websites. The websites used are listed in Table 13. The compressed version of Mnemosyne's audit logs for the entire browsing session was only 3.1 MB. This means that, on average, the disk space requirement for Mnemosyne is only .06 MB per minute for highly active browsing sessions. If we assume Mnemosyne is deployed in a typical enterprise environment, it would only require 28.8MB of storage for a single device in an 8-hour work day. If we assume a 262 workdays per year, less than 7.4GB of disk space is required to store Mnemosyne's audit log per year. For an enterprise network of 1,000 devices, only 7.4 TB of disk space is required to store the entire dataset for a single work year. This experimental evaluation shows that Mnemosyne's lightweight approach to collecting audit logs has significant improvements compared to JSGraph [43] and reduces the required storage by 82.4%.

4.7 Limitations

There are a few limitations that can occur with Mnemosyne. First, the current version of DevTools only supports attributing DOM modifications to scripts when the DOM node being inserted is an iframe or a script node. However, despite limited capability in attributing DOM modifications, Mnemosyne was able to perform exceptionally well during our experimental evaluations. This limitation was introduced because we chose not to introduce any code changes to the browser. Although, prior approaches have shown that fine-grained DOM modification attribution is feasible, it requires extensive modifications to the Blink-V8 bindings layer of the browser [43]. Since Mnemosyne was able to detect the attacks in each scenario without requiring these extensive modifications, we believe this was the correct design choice, as it clearly provides significant advantages for deployment in real-world enterprise environments. Finally, we believe that if an enterprise network prefers a more fine-grained auditing approach (e.g., using JSGraph [43]) the generated audit logs could still be leveraged by Mnemosyne's analysis modules with limited engineering effort. Second, Mnemosyne relies on a domain-versioning technique to identify the window-of-compromise. One potential limitation that could occur with domain-versioning is that the adversary could orchestrate the entire attack campaign off the compromised website. To achieve this, the adversary would need to store all malicious scripts and payloads on the compromised site's origin. If the adversary chose to use this approach, Mnemosyne's domain-versioning would not be able to identify the modifications made by the adversary. However, we argue this is extremely unlikely. First, after reviewing a corpus of over 300 well-documented sophisticated attacks carried out by various APT groups, we found that all the watering hole attacks modified the page such that it communicates with a new domain, specifically, their C&C server [2]. A main reason for this is that it provides the attacker the flexibility to update and modify the code without having to make significant modifications to the compromised website. By minimizing the modifications made, it decreases the likelihood of their attack being detected on the compromised server via the hosting organization's firewall or data loss prevention software (DLP).

Finally, as previously discussed in §3.2, Mnemosyne has limited visibility when investigating attacks that rely on a drive-by download. Specifically, Mnemosyne can only identify the "setup" phase before the browser is exploited. However, despite only recording the setup phase of a drive-by download attack campaign Mnemosyne was still capable of identifying the victims of the *driveby* attack scenario in our evaluation (§4.3.1), which demonstrates that Mnemosyne has the capability of improving the efficiency of the analysis, even when the adversary relies on a drive-by download.

5 RELATED WORK

Causality Analysis Systems. Developing systems that rely on capturing attack provenance to investigate sophisticated attacks has become a growing area of research [12, 18, 20, 25, 29, 30, 33, 39, 40, 42, 43, 48, 49, 58, 59, 61, 62, 66]. One shortcoming of whole-system provenance systems is the *dependency explosion* problem, which occurs when long-running processes communicate with many external entities. To address dependency explosion, several works have proposed partitioning the execution of a long-running process into units-of-execution [25, 42, 49, 66]. For example, BEEP [42] proposes to partition long-running processes into execution units based on the internal event loop found in applications. UIScope [66] takes a different approach and partitions the application's execution based on GUI elements of an application. However, one limitation of all existing whole-system provenance systems is the semantic gap between system level semantics and browser-level semantics. To bridge this gap, JSGraph [43] develops a customized browser that tracks fine-grained information related to the provenance graph in terms of browser-level semantics. Unlike Mnemosyne, JSGraph requires extensive modifications to the browser itself, which makes real-world deployment difficult.

Attack Detection. There has been a significant amount of work that uses attack provenance to improve the efficiency of identifying attacks in a postmortem fashion [26, 45, 52]. For example, Holmes [52] relies on the attack kill chain [65] to identify attacks. Priotracker [45] aims to improve the efficiency of postmortem analysis by automatically prioritizing abnormal causal dependencies for enterprise security. Additionally, Nodoze [23] relies on causality information to significantly reduce the number of false positives generated by industry alert systems like Splunk [9]. In addition to causality-based attack detection systems, there has also been a significant amount of work related to detecting malicious activity on the web [13, 14, 17, 37, 44, 46, 56, 57, 64, 67]. For example, Zozzle [17] detects JS-based malware by identifying syntax elements that are highly predictive of malware. Additionally, several systems have been developed that aim to detect compromised websites [13, 14, 44]. Most similar to Mnemosyne is Delta [13], which aims to identify changes associated with malicious and benign behaviors in a website. However, unlike Mnemosyne, Delta's goal is to identify compromised webpages. Mnemosyne extends this work by identify the impacts a compromised website had on an organization. that routinely visited this website.

6 CONCLUSION

In this paper, we present Mnemosyne, a novel postmortem analysis engine for analyzing sophisticated watering hole attacks. We completed an extensive evaluation on several real-world watering hole attack scenarios and our results show that Mnemosyne is capable of efficiently identifying the victims of a watering hole attack at an enterprise environment.

7 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful and informative feedback. This material was supported by National Science Foundation Graduate Research Fellowship under Grant No. DGE-1650044, and by the Office of Naval Research (ONR) under grants N00014-17-1-2895, N00014-15-1-2162 and N00014-18-1-2662. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflects the views of NSF or ONR.

REFERENCES

[1] 2015 cost of cyber crime study: United states, . http://www.ponemon.org/blog/2015-cost-of-cyber-crime-united-states.

[2] Aptnotes, . https://github.com/kbandla/APTnotes.git.

[3] dmoz. https://dmoz-odp.org/.

[4] emaildefender. https://www.secantcorp.com/emaildefender.

[5] Mitre att&ck. https://attack.mitre.org/.

[6] passivetotal. https://www.riskiq.com/products/passivetotal.

[7] How was the attack on the pfsa and polish banks carried out, and who else was targeted by criminals? https://niebezpiecznik.pl/post/jak-przeprowadzono-atak-na-knf-i-polskie-banki-oraz-kto-jeszcze-byl-na-celowniku-przestepcow/.

[8] puppeteer. https://developers.google.com/web/tools/puppeteer.

[9] splunk. https://www.splunk.com.

[10] waf. https://owasp.org/www-community/Web_Application_Firewall.

[11] F. R. Barbehenn Brittany, January 2020. https://unit42.paloaltonetworks.com/xhunt-campaign-new-watering-hole-identified-for-credential-harvesting/.

[12] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer. Trustworthy whole-system provenance for the linux kernel. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 319–334, 2015.

[13] K. Borgolte, C. Kruegel, and G. Vigna. Delta: automatic identification of unknown web-based infection campaigns. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, Oct. 2013.

[14] K. Borgolte, C. Kruegel, and G. Vigna. Meerkat: Detecting website defacements through image-based object recognition. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 595–610, 2015.

[15] K. D. Bowers, C. Hart, A. Juels, and N. Triandopoulos. Pillarbox: Combating next-generation malware with fast forward-secure logging. In *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Gothenburg, Sweden, Sept. 2014.

[16] P. Chen, L. Desmet, and C. Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

[17] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *USENIX security symposium*, pages 33–48. San Francisco, 2011.

[18] D. Devecsery, M. Chow, X. Dou, J. Flinn, and P. M. Chen. Eidetic systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Broomfield, Colorado, Oct. 2014.

[19] S. Egger, P. Reichl, T. Hoßfeld, and R. Schatz. "time is bandwidth"? narrowing the gap between subjective time perception and quality of experience. In *2012 IEEE international conference on communications (ICC)*, pages 1325–1330. IEEE, 2012.

[20] A. Gehani and D. Tariq. SPADE: support for provenance auditing in distributed environments. In *Proceedings of the 13th International Middleware Conference (Middleware)*, 2012.

[21] F. Hacquebord. Two years of pawn storm. *Trend Micro Forward-Looking Threat Research Team, April*, 25, 2017.

[22] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *Network and Distributed System Security Symposium*, 2020.

[23] 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 *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2019.

[24] W. U. Hassan, A. Bates, and D. Marino. Tactical provenance analysis for endpoint detection and response systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[25] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In *Proc. NDSS*, 2020.

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

[27] M. N. Hossain, S. Sheikhi, and R. Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *IEEE S&P*, 2020.

[28] B. S. A. Intelligence. Lazarus & watering-hole attacks, February 2017. https://baesystemsai.blogspot.com/2017/02/lazarus-watering-hole-attacks.html.

[29] 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 24rd ACM Conference on Computer and Communications Security (CCS)*, Dallas, Texas, Oct. 2017.

[30] Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee. Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1705–1722, 2018.

[31] Kaspersky. Darkhotel apt, November 2014. https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08070903/darkhotel_kl_07.11.pdf.

[32] R. P. Kasturi, Y. Sun, R. Duan, O. Alrawi, E. Asdar, V. Zhu, Y. Kwon, and B. Saltaformaggio. Tardis: Rolling back the clock on cms-targeting cyber attacks. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland), San Francisco, CA*, 2020.

[33] S. T. King and P. M. Chen. Backtracking intrusions. *ACM Transactions on Computer Systems (TOCS)*, 23(1):51–76, 2005.

[34] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2005.

[35] Y. Klijnsma. New insights into energetic bear's watering hole cyber attacks on turkish critical infrastructure, November 2017. https://www.riskiq.com/blog/labs/energetic-bear.

[36] Y. Klijnsma. Fake flash player update linked to watering hole attack on popular middle east news site, September 2017. https://www.riskiq.com/blog/labs/fake-flash-update-watering-hole-attack.

[37] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. Rozzle: De-cloaking internet malware. In *2012 IEEE Symposium on Security and Privacy*, pages 443–457. IEEE, 2012.

[38] D. P. Kwiatkowski Ivan, Aime Félix. Holy water: ongoing targeted water-holing attack in asia, March 2020. https://securelist.com/holy-water-ongoing-targeted-water-holing-attack-in-asia/96311/.

[39] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, and D. Xu. LDX: Causality inference by lightweight dual execution. In *Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Atlanta, GA, Apr. 2016.

[40] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie, et al. Mci: Modeling-based causality inference in audit logging for attack investigation. In *NDSS*, 2018.

[41] D. Lassalle, S. Koessel, and S. Abair. Oceanlotus blossoms: Mass digital surveillance and attacks targeting asean, asian nations, the media, human rights groups, and civil society, Nov. 2017. https://www.volexity.com/blog/2017/11/06/oceanlotus-blossoms-mass-digital-surveillance-and-exploitation-of-asean-nations-the-media-human-rights-and-civil-society/.

[42] K. H. Lee, X. Zhang, and D. Xu. High accuracy attack provenance via binary-based execution partition. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2013.

[43] B. Li, P. Vadrevu, K. H. Lee, R. Perdisci, J. Liu, B. Rahbarinia, K. Li, and M. Antonakakis. Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions. In *NDSS*, 2018.

[44] Z. Li, S. Alrwais, X. Wang, and E. Alowaisheq. Hunting the red fox online: Understanding and detection of mass redirect-script injections. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2014.

[45] 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 *NDSS*, 2018.

[46] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 440–450, 2010.

[47] 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 Annual Computer Security Applications Conference (ACSAC)*, 2015.

[48] S. Ma, X. Zhang, and D. Xu. ProTracer: towards practical provenance tracing by alternating between logging and tainting. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2016.

[49] 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 *Proceedings of the 25th USENIX Security Symposium (Security)*, Vancouver, BC, Canada, Aug. 2017.

[50] 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 *Proceedings of the 2018 USENIX Annual Technical Conference (ATC)*, Boston, MA, July 2018.

[51] E. A. Manzoor, S. Momeni, and L. Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, 2016.

[52] S. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. Holmes: Real-time apt detection through correlation of suspicious information flows. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2019.

[53] M. Mimoso. Apple watering hole attack, February 2013. https://threatpost.com/ios-developer-site-core-facebook-apple-watering-hole-attack-022013/77546.

[54] M. Mimoso, May 2013. https://threatpost.com/watering-hole-attack-claims-us-department-of-labor-website/100081/.

[55] M. Mimoso. Four distinct watering hole attacks dropping scanbox keylogger, October 2014. https://threatpost.com/four-distinct-watering-hole-attacks-dropping-scanbox-keylogger/109061/.

[56] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A crawler-based study of spyware in the web. In *NDSS*, volume 1, page 2, 2006.

[57] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy. Spyproxy: Execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium (Security)*, Boston, MA, Aug. 2007.

[58] C. Neasbitt, B. Li, R. Perdisci, L. Lu, K. Singh, and K. Li. Webcapsule: Towards a lightweight forensic engine for web browsers. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[59] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian. Custos: Practical tamper-evident auditing of operating systems using trusted execution. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2020.

[60] P. Paganini. Watering hole attacks on polish banks linked to lazarus group, February 2017. https://securityaffairs.co/wordpress/56235/apt/lazarus-group-polish-bank.html.

[61] P. Vadrevu, J. Liu, B. Li, B. Rahbarinia, K. H. Lee, and R. Perdisci. Enabling reconstruction of attacks on users via efficient browsing snapshots. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2017.

[62] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter. Fear and logging in the internet of things. In *Network and Distributed Systems Symposium*, 2018.

[63] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter, et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *Symposium on Network and Distributed System Security (NDSS)*, 2020.

[64] Y.-M. Wang, D. Beck, X. Jiang, and R. Roussev. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *IN NDSS*. Citeseer, 2006.

[65] T. Yadav and A. M. Rao. Technical aspects of cyber kill chain. *CoRR*, abs/1606.03184, 2016. URL http://arxiv.org/abs/1606.03184.

[66] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen. Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications. 2020.

[67] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Distributed time-aware provenance. In *Proceedings of the 38th International Conference on Very Large Data Bases (VLDB)*, volume 6, pages 49–60, Istanbul, Turkey, Sept. 2012.

[68] Z. Zorz. Banks around the world targeted in watering hole attacks, February 2017. https://www.helpnetsecurity.com/2017/02/14/banks-watering-hole-attacks/.

8 APPENDIX

8.1 Hyperparameter Optimization

The ω controls the duration of the profiling phase, which is required to learn $D_{profile}$. The proper setting of ω varies between websites, and we found that the complexity of the website and number of distinct URLs visited on the website played the largest role in determining how to set ω. To determine a reasonable value to set ω for a website, we completed an experiment that determined the number of distinct URLs required to visit in order to learn the profiling version for a website. To complete this evaluation, we relied on a subset of the Categories dataset that included 548 websites. This subset was chosen, since these sites generated a single domain version during the data collection period. Next, we measured the minimal number of distinct URL visits required to build the profiling domain version for each website category. We found that, on average, the number was relatively low, only requiring 45 distinct URLs. Therefore, the forensic analyst should set the value of ω to the average number of days necessary to observe 45 distinct URLs. We provided a detailed description of the number of distinct URLs required to visited for each category in Table 10

Namespace	Events Received
Network	responseReceived, requestWillBeSent
Page	frameAttached, frameNavigated, downloadWillBegin, windowOpen, javascriptDialogOpening
Debugger	scriptParsed
Target	targetCreated, attachedToTarget, targetInfoChanged

Table 8: The Chromium DevTools Protocol Mnemosyne relies upon to capture the necessary events to reconstruct sophisticated browser-based attacks.

	Average of Versions	Websites	Unique Pages
Alexa	2.15	1000	120,245
Categories	2.11	830	278,177
News	4.33	39	19,824
Home	3.11	38	17,581
Shopping	2.86	90	69,410
Business	2.29	31	6,770
Kids and Teens	2.24	38	11,875
Politics	2.05	148	52,977
Regional	1.98	41	15,123
Science	1.94	33	5,912
Recreation	1.78	23	3,591
Health	1.78	55	7,049
Entertainment	1.75	40	7,765
Government	1.72	67	16,286
Sports	1.68	22	6,137
Computers	1.67	64	20,519
Society	1.64	33	5,999
Art	1.54	35	6,323
Games	1.52	33	5,036

Table 9: Statistics related to the collection of data for the benign datasets.

Website Category	Unique URL Count			
	50% Coverage	75% Coverage	95% Coverage	100% Coverage
Art	7	9	29	29
Business	2	10	24	24
Computers	2	5	28	29
Entertainment	1	12	31	34
Games	1	5	24	24
Government	4	8	23	24
Health	1	4	21	21
Home	4	11	44	49
Kids and Teens	6	14	50	53
News	3	7	28	33
Politics	2	13	60	66
Recreation	1	25	41	41
Regional	4	19	36	36
Science	5	9	36	37
Shopping	3	7	57	80
Society	2	12	44	53
Sports	8	19	57	57
Average	3	10	40	45

Table 10: The average number of distinct URLs required to achieve 50%, 70%, 95%, and 100% coverage of the *profiling* domain version, $D_{profile}$.

Website	Versions	Category
hxxps://www.yomiuri.co.jp	9	News
hxxps://www.urdupoint.com	8	News
hxxps://abcnews.go.com	7	News
hxxps://www.popsugar.com	7	News
hxxps://ccm.net	7	News
hxxps://www.commentcamarche.net	7	Computer
hxxps://ameblo.jp	7	News

Table 11: The websites from Alexa Top 1k that generated more than 6 versions in one months. 6 out of 7 are News websites.

Attack Scenario	Graph Size (Nodes / Edges)	VR(s)	VP(s)	ULA(s)	Overall(s)
Malicious OAuth Access	9.2M / 16.1M	41.9	3.6	246.2	291.7
Clickjacking	4.4M / 5.9M	6.3	3.1	13.2	22.6
Malicious Software Update	628K / 1.3M	7.0	2.5	44.3	53.8
Credential Harvesting	770K / 1.6M	14.7	1.1	175.7	191.5
Keylogging	20.9M / 33.8M	105.6	14.1	756.6	876.3
Tabnabbing	6.7M / 15.0M	68.1	18.4	109.5	196.0
DriveBy	952K / 3.1M	12.1	3.4	62.5	78.0
Average		36.5	6.5	192.2	235.2

Table 12: A descriptive report of the runtime performance of each of Mnemosyne's submodules, Version Reconstruction (VR), Version Prioritization (VP), and User-level Analysis (ULA) for each attack scenario.

List of Websites Used in Storage Evaluation
cnn.com, yahoo.com, amazon.com, espn.com, foxnews.com, irs.gov, abc.com, washingtonpost.com, cbs.com, nytimes.com

Table 13: The 10 websites used for the Storage Overhead evaluation (§4.6) of Mnemosyne.

White-list of Websites Used in False Positive Analysis
cloudflare.com, cloudfront.net, doubleclick.net, facebook.net, google.com, googleadservices.com, googleapis.com, googlesyndication.com, googletagmanager.com, googletagservices.com, hotjar.com, microsoft.com, outbrain.com, twimg.com, twitter.com

Table 14: The 15 websites used for the false positive analysis of the version prioritization of Mnemosyne.

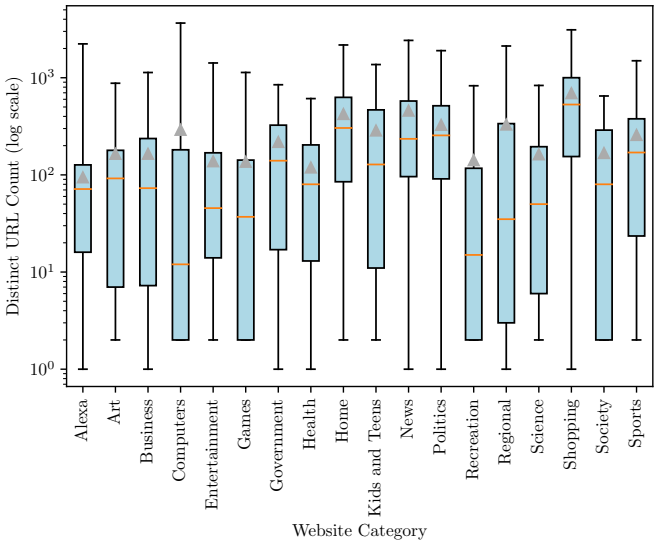

Figure 6: Five-point-summary of crawled distinct URLs for Alexa and Categories. Gray solid triangle represents mean and orange bar is median. The extreme ends are maximum and minimum, respectively.