TARDIS: Rolling Back The Clock On CMS-Targeting Cyber Attacks

Ranjita Pai Kasturi[1], Yiting Sun[1], Ruian Duan[1], Omar Alrawi[1], Ehsan Asdar[1], Victor Zhu[1], Yonghwi Kwon[2], Brendan Saltaformaggio[1]
[1]*Georgia Institute of Technology* [2]*University of Virginia*

***Abstract*—Over 55% of the world's websites run on Content Management Systems (CMS). Unfortunately, this huge user population has made CMS-based websites a high-profile target for hackers. Worse still, the vast majority of the website hosting industry has shifted to a "backup and restore" model of security, which relies on error-prone AV scanners to prompt users to roll back to a pre-infection nightly snapshot. This research had the opportunity to study these nightly backups for over 300,000 unique production websites. In doing so, we measured the attack landscape of CMS-based websites and assessed the effectiveness of the backup and restore protection scheme. To our surprise, we found that the evolution of tens of thousands of attacks exhibited clear long-lived multi-stage attack patterns. We now propose TARDIS, an automated provenance inference technique, which enables the investigation and remediation of CMS-targeting attacks *based on only the nightly backups* already being collected by website hosting companies. With the help of our industry collaborator, we applied TARDIS to the nightly backups of those 300K websites and found 20,591 attacks which lasted from 6 to 1,694 days, some of which were still yet to be detected.**

I. Introduction

Over 55% of the world's websites run on Content Management Systems (CMS) [1], with WordPress controlling nearly 60% of the CMS market [2]. Unfortunately, this widespread adoption has led to a swift increase in CMS-targeting cyber attacks. These attacks are made even easier, because CMS deployments are an amalgam of layered software and interpreters, all with varying degrees of network and system permission, which execute *on the internet-facing web server*. Worse still, this research has uncovered an unnerving trend: in-the-wild compromises of CMS deployments overwhelmingly exhibit the "low and slow" characteristics indicative of multi-stage attacks.

Despite the significant deployment of these complex software systems, to date, little research has been done to investigate and remediate CMS-targeting cyber attacks. Traditionally, the research community has turned to fine-grained logging to understand the provenance of an attack [3]–[17]. Unfortunately, in the CMS domain, these techniques are hardly deployed in practice. Specifically, despite recent advances, fine-grained logging solutions still incur notable performance/space overhead [3], [14]–[16] and often require instrumenting and training with the target systems [4], [7], [12], [17]. Moreover, website owners often have no control over the underlying web server, because the entire platform is owned and maintained by a hosting provider (e.g., HostGator [18] or even a university IT department).

For these reasons, industry standard has long shifted to a "backup and restore" model of security, offered by popular platforms such as Dropmysite [19], Codeguard [20], GoDaddy [21], Sucuri [22], and iPage [23]. Anti-virus (AV) scanners are deployed to detect compromises in websites, and nightly backups of the website's files are maintained offsite. Unfortunately, these approaches also have well-known limitations: AV signatures only catch well-known malware, they fail to detect stealthy multi-stage attacks, and high false alarm rates cause real alerts to be ignored [24], [25]. Moreover, website owners often (erroneously) revert to the most recent snapshot which did not trigger an AV alert. In fact, this research has found that website owners only take action (i.e., rollback to a snapshot) for 31% of *true alerts* and only *one-third of those* rollback to a pre-initial-infection state.

This research had the unique opportunity to study these attack trends in nightly backups from over 300,000 production websites. In collaboration with CodeGuard[1], we had initially set out to develop a website protection methodology that could replace the ineffective backup and restore standard. We began by assessing the entire history of nightly backups for 70 websites which our collaborator identified as having recently been targeted by cyber attacks. Our preliminary investigation of this dataset (detailed in §II) revealed something we had not expected: *the evolution of each attack exhibited clear multi-stage attack patterns* — slowly establishing an initial foothold, quietly maintaining persistence, lateral movement, cleaning up traces of earlier phases, etc.

Based on this discovery, we turned our attention to how forensic investigators could recover from these attacks. In order to make a practical impact in this space, we propose that forensic techniques must focus on

[1]One of the largest corporate website security and backup solutions on the market. Company name redacted for anonymous submission.

DOI 10.1109/SP40000.2020.00116

the only artifact widely available to CMS owners: *the nightly backups*. To this end, this paper presents TARDIS, a novel *provenance inference technique* which enables the investigation of multi-stage CMS-targeting attacks. Based on only the nightly backups, TARDIS reconstructs a timeline of the attack phases and recovers the *compromise window*, or the period of time during which the snapshots should not be trusted.

Through our collaboration with CodeGuard, we used TARDIS to perform a systematic study of the attack landscape across 306,830 CMS-based production websites — unique domains ranging from 38 websites within the Alexa Top 10K and 4,038 in Alexa Top 1M to mom-and-pop e-commerce sites, with nightly backups covering approximately 1900 days (March 2014 to May 2019). Based on this study, we uncovered 20,591 websites (6.7%) which were compromised with advanced multi-phase attacks. Our empirical measurement revealed several concerning facts: We found that attacks persisted in CMS websites for a minimum of 6 days and a maximum of 1694 days, with a median of 40 to 100 days. More than 20% of WordPress websites, in particular, housed attacks for over a year (likely due to WordPress's significant market share). These attacks involved stealthily dropping a huge volume of malicious code affecting the web server. We found that during an attack the number of files increased by at least 50%, ranging from visitor-attacking browser exploits to full-fledged HTML-based remote control GUIs.

II. Preliminary Investigation

Our investigation began with 70 websites that were known to have been recently compromised. We started by asking the key cyber forensics question: How would an investigator recover the website from these attacks? Unfortunately, CMS website owners generally lack the expertise and control over the hosting server required to enable robust forensic logging. Given only these nightly backups, we quickly realized that an investigator's visibility is significantly limited.

Inferring Provenance Patterns. In trying to solve this problem, we made our first key observation: A finite number of identical provenance patterns exist within the evolution of all the websites. We first found that a file in a given snapshot can exist in 1 of 3 states: added, modified, or deleted. Figure 1 illustrates the three infection scenarios we observed in the website backups. A file added (A) can be flagged as suspicious (denoted by (!)) by an AV at some point throughout its life cycle. These files could also be flagged as suspicious (by an AV) after they are modified (M). In some cases, a snapshot rollback is performed to treat the suspicious files by deleting (D) them. If the rollback deletes *all* of the attacker's files then the attack is cured, as shown in Figure 1(a). In other cases, no action is taken despite detecting a suspicious file (Uncured in Figure 1(c)).

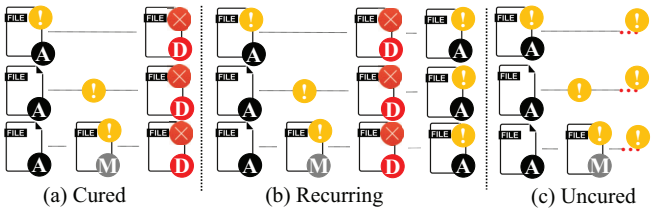

Fig. 1: Three models of temporal infection evolution.

Unfortunately, this led to the discovery that the industry standard of "backup and restore" is entirely insufficient. We found that an alarming 80% of these websites were in fact *still infected* — many website owners had rolled back to a snapshot *and patched the vulnerability*, but given their lack of forensic expertise, they were unable to identify a pre-infection snapshot (leaving initial backdoors in place and allowing the attack to *recur*).

In order to quickly rollback to a clean snapshot, investigators must recover the *compromise window*, or the period of time during which the snapshots should not be trusted. This is further complicated by the fact that each snapshot contains tens of thousands of files (11,292 on average), making this investigation a search for needles in a haystack. Not discouraged, we drilled down into the individual snapshots from a single Drupal website, `W682886`[2], which will serve as a running example throughout this paper.

Single Snapshot Metrics. When looking into the individual snapshots from `W682886`, we made our second key observation: The complexity of each snapshot can be reduced to a set of measurements, called *spatial metrics*, that highlight the existence of cyber attack evidence. In addition to the state of each file in the snapshot from before (our first spatial metric), we designed another spatial metric which measures extension mismatches among the files, i.e. if a file's internal format matches the filename's extension. Similarly, we implemented another spatial metric to identify UTF-8 based code obfuscation patterns in server-side script files. For example, in the case of `W682886`, we found *3 PHP files with obfuscated payloads disguised as icon files* in the 5 June 2018 snapshot which initiated the attack. In the end, we settled on the 9 spatial metrics detailed in §III. These spatial metrics were effective at highlighting the presence of cyber attack artifacts within a single snapshot. However, while this was a good first step, it was neither sufficient to explain the evolution of the attack nor to understand the length of compromise.

Temporal Evolution Of Attack Phases. We collected spatial metrics to represent each snapshot of `W682886`, paying specific attention to sudden changes between pairs of consecutive snapshots. This revealed our third key observation: Modelling the *implicit events* which trigger these sudden changes can expose the attack

[2]Website domain is omitted pending responsible disclosure.

TABLE I: Temporal File Differential Analysis.

Date	Outlier	PHP	HTML	ASCII	XML	PNG	ZIP
20 Apr	-	0	0	+1	0	0	0
21 Apr	!	+7	+1	+3	+21	0	0
22 Apr	-	-2	0	+1	0	0	0
23 Apr	-	0	+2	+2	0	0	0
24 Apr	-	0	0	+1	0	0	0
25 Apr	-	+3	-1	+6	0	0	0
05 Jun	!	-13	+5	+50	0	+9	+1
07 Jun	!	-31	0	+1	0	0	0
08 Jun	!	-18	-6	-22	-20	-9	-1
09 Jun	-	+5	0	0	0	0	0
10 Jun	-	0	0	+3	0	0	0
11 Jun	-	+5	0	+1	0	0	0
12 Jun	-	+3	-7	-4	0	0	0
13 Jun	!	+9	+13	+28	+20	0	+1
14 Jun	!	-13	-13	-26	-20	0	-1
15 Jun	-	0	0	+1	0	0	0
16 Jun	-	0	0	+1	0	0	0

phases. This led us to plot the temporal progression of the spatial metrics across all of `W682886`'s snapshots.

Table I shows one such progression considering only a single spatial metric, i.e. the file format numbers. The temporal evolution of this metric exposed the first attack signs. As seen in Table I, sudden changes in the file format metric stand out on 21 April, 5-8 June, and 13-14 June. We found identical spatial metric outliers in 3 other Drupal websites from 14 April to 21 May 2018, suggesting the attack's lateral movement. Canali et.al. [26] also found that web attacks dropped large volumes of files on the web server, which explains the sudden changes we observed in the file format metrics. We also observed that these patterns evolved similarly over time — adding more functionality to the existing malicious code (e.g. it started with only file read capabilities, and after 8 days evolved to modify files and communicate over an SSL gateway). Eventually, we saw that these attacks tried to clean up their footprints by deleting most of the attack files.

Attack Model. These patterns formed the basis of the multi-stage attack model presented in this paper. Our study found that these attacks consisted of slow and steady attack patterns starting with establishing an initial foothold, malware injection, maintaining persistence, lateral movement, and eventually cleaning up any traces of malicious activity. This was confirmed by our case studies (§VI), which provide an intriguing view of this widespread attack evolution.

Taken together, the above key observations drove our design of TARDIS. Modeling the temporal evolution of the spatial metrics allows TARDIS to infer the provenance of attack evidence. Further, identifying outliers within that evolution reveals both the compromise window (starting Apr 21 for `W682886`) and the progression of the attack phases. Using TARDIS, forensic investigators know where to focus their efforts and website owners can quickly revert the website to a clean snapshot. In §IV, we will revisit these original 70 websites as manually-investigated ground truth to evaluate the effectiveness of the TARDIS framework.

III. Design

TARDIS overcomes the challenges described in §II via a novel provenance inference technique, using only the nightly backups of the CMS deployment. Figure 2 shows the phases of TARDIS's operation: First, TARDIS constructs a temporally ordered set of spatial elements from each snapshot (§III-A). It then computes spatial metrics for each individual snapshot's elements (§III-B). This is followed by temporally correlating the collected spatial metrics and querying them against *attack models* to recover the timeline and label attack events (§III-C). Finally, it verifies the sequence of assigned attack labels and extracts the compromise window (§III-D).

A. Spatial Element Sequencing

TARDIS extracts the files associated with each night's snapshot and maps them as spatial elements $(el_j\,(\psi_i) \in \mathbb{V}_i)$ for each snapshot $\psi_i \in \Psi$. Here, Ψ is the set of all ψ_i, the label i denotes the index of the temporal snapshot under analysis, and j denotes the index of a spatial element in $\mathbb{V}_i$. Basically, ψ_i is a point in time when the i^{th} snapshot was taken. $\mathbb{V}_i$ is the set of spatial elements (el_j) collected at time ψ_i. For example, the initial snapshot is collected at ψ_0, the next snapshot at ψ_1 and so on. At snapshot ψ_0, the set of elements are represented as $\mathbb{V}_0 = [el_0, el_1, ...]$. These elements $(el_j\,(\psi_i)) \in \mathbb{V}_i$ reside in the space Θ that denotes the monitoring *space* of all spatial elements (i.e., all versions of all files hosted on the web server).

While processing each temporal snapshot ψ_i, a set of initial spatial metrics $(m_k\,(\psi_i) \in \mathbb{M}_i)$ are recorded in the set $\mathbb{M}_i$. Here, the label k denotes the index of the spatial metrics collected at temporal snapshot ψ_i. These initial spatial metrics consists of the file type counts, and the state of each spatial element in terms of added, modified, or deleted. $\mathbb{M}_i$ is further populated with carefully selected measurements as discussed in §III-B. A comprehensive definition of the terminology used is presented in Table II.

For example, the website `W682886`'s initial snapshot (ψ_0) contains $11,327$ files. All of these files are mapped as a sequence of spatial elements in $\mathbb{V}_0$. As an example of a single spatial metric, this snapshot also contains 23 different file types (e.g. PHP, HTML, JS, CSS, etc.). This information is recorded within the spatial metric set $\mathbb{M}_0$. If the backups are collected on a nightly basis for 3 months (e.g., 91 backups), then:

$$\mathbb{V} = [\mathbb{V}_0, \mathbb{V}_1, ..., \mathbb{V}_{90}]$$
$$\mathbb{M} = [\mathbb{M}_0, \mathbb{M}_1, ..., \mathbb{M}_{90}]$$
$$\mathbb{V}_0 = [el_0, el_1, ..., el_{11326}]$$
$$\mathbb{M}_0 = [num\,(PHP) = 727, num\,(CSS) = 829, ...]$$

B. Spatial Analysis

The set of spatial elements comprise of various file types (such as PHP, HTML, JS, CSS, images, plaintext, etc.), each of which requires disparate investigation techniques to identify attack attributes. To address this challenge, we

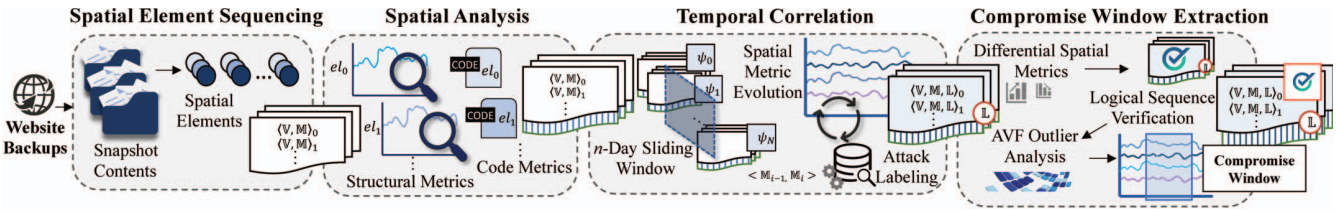

Fig. 2: TARDIS Overview. Phase 1 constructs spatial element sets from the website backup. Phase 2 computes the structural and code metrics for each individual snapshot. Phase 3 temporally correlates the collected metrics and labels attack events. Phase 4 verifies the assigned attack labels and extracts the compromise window.

split spatial analysis to extract two types of metrics: (1) structural metrics and (2) code metrics.

1) Structural Metrics: With the computed set of spatial elements $\mathbb{V}$ and the initial metrics $\mathbb{M}$ for each temporal snapshot, we turn to investigating this set $\mathbb{V}$. Based on our observations from the preliminary study, we developed a suite of lightweight measurements that highlight the existence of suspicious elements.

Hidden Files and Directories. Long-lived multi-stage attacks can be characterized by the attacker's intent to modify the existing setup and laying low at the same time. During our preliminary study, we observed that this was achieved by dropping malicious and/or suspicious elements as a hidden file or by placing them in a hidden directory to evade first order defenses. TARDIS employs pattern matching by filtering the typically expected hidden elements (such as .htaccess) and appends a structural metric $Hide(el_j(\psi_i))$ to $\mathbb{M}_i$ upon finding an element $el_j \in \mathbb{V}_i$ in a hidden location, because websites did not often employ hidden files or directories.

Extension Mismatch. We also observed that another common tactic used in CMS-targeting attacks was to disguise a server-side executable as something else. For example, we commonly observed spatial elements renamed deceptively as an icon file (e.g. favicon.ico) but containing PHP code to evade less technical CMS users. TARDIS uses the spatial element's filename to extract its extension and then matches it against the inferred file format (e.g., via the file type's magic number or other formatting that can identify the type of file). If TARDIS finds a discrepancy while matching the file type and the file extension for an element $el_j \in \mathbb{V}_i$, it appends the structural metric $ExtMis(el_j(\psi_i))$ to $\mathbb{M}_i$.

Filename Entropy. Another indicator of suspicious activity seen in CMS-targeting attacks is *long, incoherent, or randomly generated filenames*. TARDIS measures the entropy of filenames for all spatial elements el_j. A higher entropy indicates a more random filename that is less likely to be a human-generated benign filename. Entropy is measured by password strength calculation logic [27], which computes a filename's "randomness" score by measuring its similarity to several dictionaries, spatial keyboard patterns (e.g., QWERTY, Dvorak), repetition of a single character, sequences of numbers or characters, and other commonly used keywords (e.g., `133t`). For TARDIS, the password strength output was analogous to higher entropy (more randomness) and thus a more suspicious filename.

Since it is not possible to identify an absolute threshold for high entropy in filenames, TARDIS compares the relative entropy of the spatial elements using the median absolute deviation (MAD [28]) test. Specifically, instead of computing an absolute threshold for filename entropy, which is difficult to predict with certainty, TARDIS considers all the elements in a given temporal snapshot to first find the median entropy of all elements, followed by computing the median absolute deviations for each element and eventually checking if the median absolute deviation is greater than a relative threshold. When a relatively higher entropy is identified for an element $el_j \in \mathbb{V}_i$ from a temporal snapshot ψ_i, the structural metric $HEntrp(el_j(\psi_i))$ is appended to $\mathbb{M}_i$.

Permission Change. TARDIS uses temporal tracking of each spatial element to detect permission changes between snapshots. In particular, when the permissions

TABLE II: Formal Definitions of the State of the CMS Deployment.

Name	Symbol	Definition	Description
Time	$\Psi = (\psi, ...)$	$\Psi = (\mathbb{Z}, +)$	Time measured in terms of the snapshot versions.
Space	$\Theta = (\theta, ...)$	$\Theta = (\mathbb{Z}, +)$	Space of *elements* that can be monitored.
Elements	$\mathbb{V} = (el, ...)$	$el = el\,(\theta, \psi, \psi')$	Files under investigation within their life span.
Spatial Metrics	$\mathbb{M} = (m, ...)$	$m = m\,(\theta, \psi)$	Measurements computed against a single night's snapshot of the website backup attributes.
Labels	$\mathbb{L} = (lb, ...)$	$lb = lb\,(\psi, \theta)$	An enumerable set of labels describing the *events* associated with the security of the *elements*.

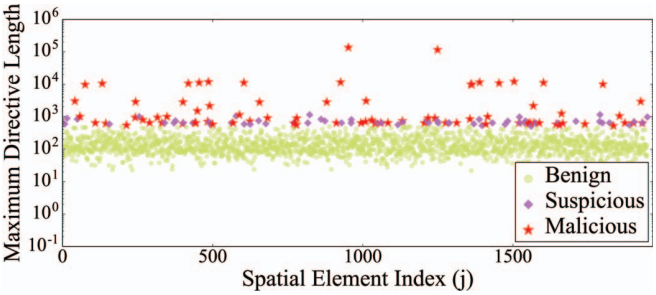

Fig. 3: Outlier detection within the directive length distribution of all code elements in one snapshot.

of spatial elements change from non-executable (read-only, read-write, etc.) to executable, it raises suspicion since it is unusual for a developer to start with a non-executable and provide execute privileges to it. An observation from our study was that multi-stage attacks package shell scripts in a text file and then change the permissions of the file to explore privilege escalation opportunities. Upon identifying an element $el_j \in \mathbb{V}_i$ from a temporal snapshot ψ_i with permission change equipping it with execute capabilities, TARDIS appends a structural metric $Exec(el_j(\psi_i))$ to $\mathbb{M}_i$.

2) Code Metrics: Since we are interested in the investigation of server-side attacks targeting CMSs, TARDIS analyzes the spatial elements containing code. These collected metrics are recorded for each snapshot ψ_i and appended to the spatial metric set $\mathbb{M}$.

Script Directive Outlier Analysis. Most of the server-side source code is either part of the CMS core, associated plugins, or website-owner developed code. As they are meant to be maintained by developers, it is unusual to find source code files among the spatial elements with script directives (parsable instruction sequences) that are thousands of characters long. Hence, we observed that injecting exceptionally long and complex lines of obscure code in the spatial elements is a strong hint that can be leveraged to identify attack behaviors. Our study found that attackers use this tactic to limit the readability of injected code, delaying immediate reverse engineering attempts.

Figure 3 shows the directive length distribution for all spatial elements containing server-side code for `W682886`'s 2 May 2018 snapshot. The x-axis presents the spatial element index j, and the longest directive length for each of these code files is plotted along the y-axis. In benign elements (green dot) none of the directives were more than 500 characters long, whereas most attacker-injected elements (red star) in this snapshot contained directives longer than 1500 characters. There was a mix of benign and malicious elements with maximum directive length between 500 and 1500 characters, which becomes the suspicious range (purple diamond).

Despite learning that long directives in spatial elements are suspicious, finding a threshold for directive length is not feasible due to varied coding styles and practices followed by different developers. However, it is possible to decide if a spatial element is suspicious by relatively comparing all the elements in any given temporal snapshot and performing outlier analysis. We leverage this observation to find suspicious files with relatively long directives using the median absolute deviation (MAD) previously described in §III-B1. Upon detection of the suspiciously long directive lines in a spatial element $el_j \in \mathbb{V}_i$ from a temporal snapshot ψ_i, TARDIS appends the code metric $LongLine(el_j(\psi_i))$ to the spatial metric set $\mathbb{M}_i$.

Obfuscation Detection. We observe that server-side malware often uses a string that contains both UTF-8 characters (i.e., wide characters) and traditional 8-bit characters. While the construction of such a string itself is not malicious, it is a commonly used tactic to avoid detectors that look for known malicious string/code snippets. For example, the malicious PHP file disguised as an icon file which we mentioned earlier is included from the root of the CMS using the following *long UTF-8 (black) coupled with ASCII (red) path* to the file:

```
@include "\x2fmn\x74/s\x74or\x31-w\x632-\x64fw\x31/4\
    x3505\x327/\x77ww\x2ecv\x6dar\x61ci\x6eg.\x63om\
    x2fwe\x62/c\x6fnt\x65nt\x2fmo\x64ul\x65s/\x61gg\
    x72eg\x61to\x72/t\x65st\x73/f\x61vi\x63on\x5fbd\
    x33fd\x35.i\x63o";
```

Array map obfuscation is another obfuscation scheme commonly used to evade defenses [26]. An array map is defined to map each character to a different character. This map is used to deobfuscate what appears to be a jumbled list of characters to a reverse engineer trying to make sense of this obfuscated spatial element. For example, in the following code snippet, `lnhqvwxeon()` is a function that takes a jumbled character string (in the variable `$zvkgw`) and uses the array map in `$lyfuf` to generate malicious code that gets executed as part of the PHP `eval` function:

```
$lyfuf = Array('1'=>'G', '0'=>'6', '3'=>'4',
    '2'=>'L', '5'=>'1', '4'=>'W', '7'=>'y', ... ,
    'y'=>'w', 'x'=>'F', 'z'=>'l');
eval(lnhqvwxeon($zvkgw,$lyfuf));
```

Upon spatial detection of obfuscation in an element $el_j \in \mathbb{V}_i$ from a temporal snapshot ψ_i via regex pattern matching for the cases described above, TARDIS appends a code metric $Obfus(el_j(\psi_i))$ to $\mathbb{M}_i$ indicating the presence of obfuscation in the element el_j.

Suspicious Payload Evaluation. In server-side spatial elements, functions such as `eval`, `base64_decode`, and `url_decode` are commonly paired to execute previously identified obfuscated code. TARDIS identifies and flags instances of the `eval` and `base64_decode`/`url_decode` pairing via pattern matching along each control flow. Upon identifying this code unwrapping technique in an element $el_j \in \mathbb{V}_i$ from a

TABLE III: Rules to Model Compromised CMS Events as Multi-Stage Attack Phases.

Attack Label $\mathbb{L}$	Severity	Attack Modeling Rule
Establish Foothold	Medium	$ExtMis(el_j(\psi_i)) \vee [(el_j \notin \mathbb{V}) \wedge [HEntrp(el_j(\psi_i)) \vee Hide(el_j(\psi_i))]]$
Obfuscated Code Injection	High	$[(size(el_j(\psi_i)) > size(el_j(\psi_{i-1}))) \vee (MaxL(el_j(\psi_i)) > MaxL(el_j(\psi_{i-1})))] \wedge Obfus(el_j(\psi_i))$
Malware Dropped	High	$(el_j \notin \mathbb{V}_{i-1}) \wedge [Obfus(el_j(\psi_i)) \vee LongLine(el_j(\psi_i)) \vee EvDc(el_j(\psi_i))]$
Code Generation Capability	Low	$CodeGen(el_j(\psi_i))$
Defense Evasion	High	$Hide(el_j(\psi_i)) \wedge [Obfus(el_j(\psi_i)) \vee EvDc(el_j(\psi_i)) \vee HEntrp(el_j(\psi_i)) \vee ExtMis(el_j(\psi_i))]$
Escalate Privileges	High	$Exec(el_j(\psi_i)) \wedge \neg Exec(el_j(\psi_{i-1}))$
Maintain Presence	Medium	$(Sev(el_j(\psi_i)) == High) \wedge (Sev(el_j(\psi_{i-1})) == High)$
Attack Cleanup	Medium	$(Sev(el_j(\psi_{i-1})) == High) \wedge [(Sev(el_j(\psi_i)) == None) \vee (Sev(el_j(\psi_i)) == Low) \vee ((el_j(\psi_i)) \notin \mathbb{V}_i)]$

temporal snapshot ψ_i, TARDIS appends a code metric $EvDc(el_j(\psi_i))$ to $\mathbb{M}_i$ indicating unsafe or suspicious code, compressed to avoid more conventional detectors.

Code Generation Capability. We observed that almost every server-side spatial element contributing to the multi-stage CMS-targeting attack contained code generation capabilities such as the use of `create_function`. Although several developers use this as part of certain CMS plugins, it is very rarely employed in ordinary server-side code development. TARDIS scouts for such code generation capabilities and appends a code metric $CodeGen(el_j(\psi_i))$ to the spatial metric set $\mathbb{M}_i$ upon finding an element $el_j \in \mathbb{V}_i$ satisfying the constraints.

C. Temporal Correlation and Forensic Recovery

Based on the collected spatial metrics for each snapshot, TARDIS now attempts to temporally correlate these metrics across snapshots to identify suspicious activities that evolve within the website. Here, TARDIS is programmed to track developments over a sliding $n - day$ time window (e.g. $n = 20$ means track developments in the spatial metrics by comparing them across 20 days). In this stage, TARDIS temporally correlates the spatial metric set $\mathbb{M}_i$ at any temporal snapshot ψ_i with the spatial metrics $\mathbb{M}_x$ from all previous temporal snapshots within the sliding window $(i - n < x \leq i)$ to capture the persistent adversary relationship and extract the timeline of events.

Patterns in the metrics $\mathbb{M}$, assigned as a function of spatial elements, are indicative of long-lived multi-stage attack behaviors which can be detected. We construct rules to encode these behaviors based on the Boolean composition of the spatial metrics. These rules are designed to be agnostic to the individual metrics and are based on the invariants of the phases that long-lived multi-stage attacks go through. Table III shows the representative set of rules applied as part of the current implementation. Further, the temporal correlation of events encapsulating the patterns in spatial metrics is implemented by considering two consecutive temporal snapshots at a time. In particular, the 2-tuple $\langle \mathbb{M}_{i-1}, \mathbb{M}_i \rangle$ is passed to TARDIS's temporal correlation phase (as shown in Figure 2) where it is queried against the attack models from Table III. An attack label set $\mathbb{L}_i$ and a severity are assigned to each temporal snapshot, thus incrementally building the attack timeline. The assigned severity of the attack labels tells the investigator which of the labels are more critical than the others.

The rules presented in Table III capture the overall intuition behind our insights. For example, our running example `W682886` has two cases of obfuscated code injection: (1) Suspicious obfuscated code injected into an existing unobfuscated element. (2) Additional obfuscated code appended to an already obfuscated element. Based on this observation, if an obfuscated spatial element $el_j(\psi_i) \in \mathbb{V}_i$ increases in size (i.e. obfuscated attack progression), or if a script directive outlier is flagged in $el_j(\psi_i)$ but not $el_j(\psi_{i-1})$ (i.e. obfuscated code is injected into an existing unobfuscated element), and the code metric $Obfus(el_j(\psi_i)) \in \mathbb{M}_i$, then an attack label "Obfuscated Code Injection" is appended to the set $\mathbb{L}_i$ at snapshot ψ_i. For `W682886`, we see this label assigned on 21 April, 7 June, and 13 June 2018.

Note that multiple spatial elements $el_j(\psi_i) \in \mathbb{V}_i$ can give rise to multiple labels for each temporal snapshot. For example, there can be three spatial elements associated with $Obfus(el_j(\psi_i)) \in \mathbb{M}_i$ (i.e. 3 files with obfuscated code in them), and four other spatial elements associated with $ExtMis(el_j(\psi_i)) \in \mathbb{M}_i$ (i.e. four shell scripts disguised as gifs). In this case, both event labels *Obfuscated Code Injection* and *Privilege Escalation* are appended to the set $\mathbb{L}_i$, and the highest severity of the union of this set $\mathbb{L}_i$ is assigned to the temporal snapshot ψ_i. It is also possible that multiple labels get assigned to a temporal snapshot due to one spatial element, i.e. an adversary can move a benign file to a hidden directory and inject it with suspicious obfuscated code. In this case, both *Defense Evasion* and *Obfuscated Code Injection* labels get appended to the set $\mathbb{L}_i$, and follow the highest severity assignment as described earlier.

D. Compromise Window Recovery

With the attack labels in hand, TARDIS proceeds to extract the compromise window by parsing consecutive pairs of the 3-tuple of spatial elements, spatial metrics,

Algorithm 1: Compromise Window Detection

Input: $\mathbb{V} = [\mathbb{V}_0, \mathbb{V}_1, ..., \mathbb{V}_{N-1}]$, $\mathbb{M} = [\mathbb{M}_0, \mathbb{M}_1, ..., \mathbb{M}_{N-1}]$, $\mathbb{L} = [\mathbb{L}_0, \mathbb{L}_1, ..., \mathbb{L}_{N-1}]$, $N = Number\ of\ temporal\ snapshots$
Output: $SuspiciousRanks = [\psi_{x_0}, \psi_{x_1}, ..., \psi_{x_{N-1}}]$, $CompromiseWindow = [\psi_{x_0}, \psi_{x_1}, ..., \psi_{x_k}]$

`// Calculate frequency of each attribute value`
1 **for** $\forall \psi_i \in \Psi$ **do**
2 $DiffAttr_i \leftarrow \mathbb{V}_i - \mathbb{V}_{i-1}$, for each $el_j \in \mathbb{V}_i$
3 $DiffAttr_i \leftarrow \mathbb{M}_i - \mathbb{M}_{i-1}$, for each $m_j \in \mathbb{M}_i$
  `// Verify label sequence`
4 **if** $i! = 0$ *and* $\mathbb{L}_i$ *comes after* $\mathbb{L}_{i-1}$ **then**
5 $CorrectLabel_i = True$
6 **end**
7 **end**
8 $AttrFreq$ = Frequency of each attribute $da_j \in DiffAttr$
`// Calculate AVF scores`
9 **for** $\forall DiffAttr_i \in DiffAttr$ **do**
10 $score \leftarrow 0$;
11 **for** $da_j \in DiffAttr_i$ **do**
  `// Score for snapshot` ψ_i
12 $score \leftarrow score + AttrFreq[da_j]$
13 **end**
14 $AVFscores \leftarrow score/size(DiffAttr_i)$
15 **end**
16 $SuspiciousRanks \leftarrow$ return (sort ψ_i in order of minimum $AVFscores$)
17 **for** $\forall \psi_i \in \Psi$ **do**
18 **if** $CorrectLabel_i == True$ **then**
19 **while** $AVFScores\ outside\ CompromiseWindow <$ $AVFScores\ inside\ CompromiseWindow$ **do**
20 $CompromiseWindow \leftarrow$ compute (range between first and last ψ_i with verified $\mathbb{L}_i$)
21 **end**
22 **end**
23 **end**
24 return $SuspiciousRanks$, $CompromiseWindow$

and the assigned attack labels (i.e. $\langle \mathbb{V}, \mathbb{M}, \mathbb{L} \rangle_i$). Algorithm 1 presents the pseudocode for this procedure. Lines 1-3 in Algorithm 1 describe how it takes the 3-tuple $\langle \mathbb{V}, \mathbb{M}, \mathbb{L} \rangle$ as input, computes the differential spatial metrics $DiffAttr_i$ for each snapshot at ψ_i from consecutive pairs of $\langle \mathbb{V}, \mathbb{M} \rangle_{i-1}$,$\langle \mathbb{V}, \mathbb{M} \rangle_i$ (e.g. recall the differential file type information shown in Table I).

As shown in Lines 8-16 in Algorithm 1, it then computes the attribute value frequencies (AVF) using the AVF algorithm [29] on the differential spatial metrics $DiffAttr_i$ and processes it to rank the temporal snapshots ψ_i in order of suspicious activities. The AVF algorithm performs well on categorical data with multiple attributes, the differential spatial metrics in our case [29]. In a typical AVF application, the number of anomalies to be detected are pre-programmed. Here, instead of choosing the number of anomalies to be detected, TARDIS uses the AVF algorithm to rank the temporal snapshots in the compromise window in the order of most suspicious to least suspicious.

Before TARDIS outputs the final attack labels for the entire temporal sequence, it passes the label set $\mathbb{L}$ through *logical sequence verification* of the associated labels (Lines 4-6 in Algorithm 1) and assesses their order of appearance. For example, when the only labels assigned are 'code generation capability' and 'attack cleanup', it has been observed that these behaviors arise from benign elements populated by the web developer and mean no harm. In such cases, the labels are retained but their severities are reduced to 'None'. If the label 'maintain presence' is seen on a snapshot prior to any other event label such as 'establish foothold' or 'malware dropped' or any other high severity modeling rule, since we know that this event sequence is intuitively not feasible, TARDIS has been programmed (again via Boolean composition of the previous label rules) to filter out sequences that do not make logical sense.

Notice that TARDIS's compromise window is only influenced by the order of 2 out of the 8 labels, i.e. *attack cleanup* and *maintain presence*. TARDIS considers all combinations of the other labels as the beginning of a compromise window. This makes TARDIS robust against attackers who might try to deploy out-of-order payloads to confound TARDIS.

Once the logical sequence of the assigned labels is verified and the temporal snapshots are ranked in the order of suspicious activities, TARDIS then identifies the *compromise window* — the period between the first and the last temporal snapshot comprising of suspicious activities with assigned and verified labels $\mathbb{L}$. Also, the window is chosen such that the AVF score for every temporal snapshot outside the compromise window is higher than the score for every temporal snapshot within the compromise window (lines 17 - 21 Algorithm 1). This is the period when maximum suspicious activities take place in the website and help the investigator narrow down the analysis to a smaller window. We find that these intuitive temporally correlated spatial metrics and the attack models both align well with the design and work well in practice, as we show in §IV and §V.

For our website under investigation `W682886`, from 1 April - 30 June 2018, the compromise window is identified from 21 April - 16 June 2018. By applying the AVF algorithm, TARDIS outputs the following temporal snapshots for this website ranked in order of most suspicious to least suspicious as follows:

```
<- Most suspicious.............Least suspicious ->
5 June, 13 June, 8 June, 14 June,21 April,..,29 June
```

This aligns with our earlier visual inspection of the differential file type metrics presented in Table I.

Note than these attack models are scalable irrespective of the underlying CMS, i.e. when a new tactic is identified, the TARDIS framework is designed to be highly modularized and can be easily updated to capture the essence of the new tactic and the attack label associated with it. Essentially, applying the attack modeling rules to spatial metrics and incrementally sliding along each temporal snapshot enables TARDIS to assign appropriate labels $\mathbb{L}$ along the compromise window, thus providing a timeline of the events as part of the long-lived multi-stage attack investigation.

IV. Validating our Intuition

This research began with the key insight that CMS-targeting cyber attacks exhibit the "low and slow" characteristics indicative of multi-stage attacks. Based on this, we designed TARDIS to recover the compromise window and reconstruct the attack timeline. We now perform several micro-benchmarks with a ground truth set of websites to validate this intuition.

Data Set and Ground Truth. Our preliminary study in §II looked at the nightly backups of 70 CMS websites which CodeGuard had identified as recently compromised. We manually investigated these websites and labeled the observed attack models. We will use these 70 websites again here as ground truth. To these 70 websites, we added the full history of nightly backups from 93 additional CMS websites, selected randomly from our collaborator's data. Again, we performed a manual investigation of these 93 new websites to obtain ground truth (discussed below). This yielded a total of 163 websites, each of which represents backups collected nightly during a 13-month period between April 2018 and May 2019.

In order to validate TARDIS's performance, we manually investigated the 163 websites to obtain ground truth. We first installed a clean version of the CMS locally and removed any file which had not been modified for each snapshot. We then searched all the code files for malware payloads and confirmed our findings with CodeGuard engineers. If our investigation found an attack, we labeled the snapshot when the attack first appeared. If the attack was cleaned up via a rollback, we labeled the corresponding snapshot. We then annotated the expected attack labels for every snapshot within that compromise window. From our 163 websites, 80 were found to be compromised. We note that this is biased by the original 70 (all of which were known-compromised) but still provides a varied test suite of benign and malicious cases.

We identified the CMS platform used by each website using WhatCMS [30] and CMS Garden [31]. The CMS market share distribution in this dataset is shown in Table IV. The distribution is roughly similar to the real-world distribution of CMS-based websites, e.g., a majority of websites are built on WordPress with Drupal and Joomla a close second and third.

A. Identification of Attack Models

We then drilled down into the websites identified as compromised. To validate TARDIS's attack timeline reconstruction capability, we first ran each website's sequence of backups through TARDIS and recorded what attack labels were assigned to each nightly snapshot. Note that TARDIS did not have nor need access to our ground truth for investigating the website backups, and it relied only on the temporal correlation of spatial metrics and attack models for timeline extraction.

TABLE IV: Distribution of Compromises in the Evaluation Dataset of 163 Websites.

CMS	# of websites[1]	#GT[2]	TARDIS [3]		
			#TP[4]	#FP[5]	#FN[6]
WordPress	92	47	47	1	0
Drupal	23	15	15	1	0
Joomla	17	10	10	0	0
PivotX	9	2	2	0	0
Prestastop	2	0	0	0	0
TYPO3 CMS	8	3	3	1	0
Bourbon	4	1	1	0	0
Contao	3	0	0	0	0
Contenido	5	2	2	0	0
Total	163	80	80	3	0

1: Total number of websites evaluated for each CMS.
2: Total number of compromised websites (Groud Truth)
3: Total number of websites flagged as compromised by TARDIS.
4: True Positive, 5: False Positive, 6: False Negative.

We then compared the TARDIS output attack labels with our manually derived ground truth.

Table V presents the micro-benchmark results for the 163 websites. The CMS platform is listed in Column 1. For each CMS platform, the subsequent pairs of columns show the number of websites which TARDIS marked as containing each attack label (denoted by #) and the number of those labels which were false positive cases (denoted by #FP), i.e. our derived ground truth for that website does not contain that attack label. For example, Row 3 of Table V shows TARDIS labeled obfuscated code injection in 8 Joomla-based websites and 1 of them is an FP. Note also that any attack labels detected in known clean websites were marked as FPs.

From Table V, we make several observations: Taken individually, TARDIS's attack models detect the attack labels within compromised websites with high accuracy. The labels escalate privileges and establish foothold were identified with zero FPs, as seen in Table V. Also, obfuscated code injection, maintain presence, malware dropped, and attack cleanup labels saw low FP counts of 1, 1, 3, and 4, respectively, highlighting TARDIS's attack model detection accuracy.

Most importantly, when all of these attack models are considered together within a single website, TARDIS is able to prune individual false positives. TARDIS verifies the logical sequence of the recorded attack labels, computes the compromise window, and then tags the website as compromised or not, as discussed §III-C. This procedure pruned 38 of the 39 FPs in the code generation capability label, all 4 FPs in attack cleanup, all 3 FPs in malware dropped, 8 of the 9 FPs in defense evasion, and the only FP in obfuscated code injection, effectively removing 94.7% of the FPs listed in Table V.

Another observation is that the attack tactics vary greatly across CMS platforms, but a few labels are present in *all attacks*. In particular, the maintain presence, malware dropping, and defense evasion labels are seen in all compromised CMSs. This is confirmed by our ground truth investigation. This may seem intuitive, but it confirms our premonition that *CMS-targeting*

TABLE V: Evaluation of the Multi-Stage Attack Phase Models.

CMS	Obf. Code Injection		Maintain Presence		Code Gen. Capability		Malware Dropped		Attack Cleanup		Establish Foothold		Defense Evasion		Escalate Privileges	
	#	#FP	#	#FP	#	#FP	#	#FP	#	#FP	#	#FP	#	#FP	#	#FP
WordPress	17	0	39	1	71	24	46	2	21	4	2	0	42	6	4	0
Drupal	6	0	15	0	18	4	13	0	9	0	0	0	12	2	5	0
Joomla	8	1	8	0	13	4	7	1	2	0	0	0	10	1	2	0
PivotX	1	0	2	0	2	0	1	0	0	0	0	0	2	0	0	0
Prestastop	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
TYPO3 CMS	0	0	2	0	3	1	3	0	1	0	0	0	2	0	0	0
Bourbon	0	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0
Contao	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Contenido	2	0	1	0	5	3	2	0	1	0	0	0	2	0	1	0

attacks overwhelmingly exhibit long-lived multi-stage attack behaviors.

Notice that TARDIS recorded 115 out of the 163 websites with code generation capabilities (a common tactic used in multi-stage attacks). However, from the Column #FP for code generation capability label in Table V, we find that in 39 of the websites, these labels were FPs. This can be attributed to the open-source nature of the CMSs and the varied coding practices followed by CMS plugin developers. The other significantly higher FP (9 out of 71 websites) is defense evasion; these are due to the presence of obfuscation used to prevent visibility into paid CMS plugins that appear like multi-stage attack behaviors at first glance.

B. Multi-Stage Attack Timeline

Based on our previous runs of TARDIS, we recorded the compromise window which was reported or "Not Compromised" for each website. We evaluate TARDIS's correctness by comparing these with our manually recorded compromise labels for the 163 websites. The results are presented in Table IV.

Table IV shows the CMS platform and its distribution in Columns 1 and 2, respectively. Column 3 (#GT) presents the ground truth number of compromised websites in this dataset derived by manual investigation. Columns 4 through 6 show the number of websites for which TARDIS output a compromise window. Column 4 (#TP) presents the number of websites for which the TARDIS compromise window output matched the ground truth, and Column 5 (#FP) presents the number of false positives produced by TARDIS. Here, FP is essentially a "false alarm", meaning that TARDIS produced a compromise window, but the website was known to not be compromised (via our ground truth). Column 6 (#FN) presents the number of websites that are compromised and not identified by TARDIS.

Overall, TARDIS found a total of 83 websites infected with multi-stage attacks. Interestingly, more than 50% of these attacks targeted WordPress CMS, as seen in Table IV. In addition to the 70 known-compromised websites from §II, TARDIS found the 10 additional attacks in the added set of 93 websites. TARDIS reported an attack timeline that matched our ground truth for these 80 compromised websites. Manual verification confirmed the correctness of this result. To the best of our knowledge, we did not find any websites that contained an attack that was missed by TARDIS, thus showing a zero FN count.

Notice that TARDIS produced 3 FPs, i.e. Column 4 from Table IV shows 3 websites (one from WordPress, Drupal, and TYPO3 CMS). Our manual investigation revealed that all 3 websites contained user-developed security plugins with obfuscated code, similar to the tactic used by attackers, which caused TARDIS to output a compromise window for these websites. Note that there are several publicly available security plugins that contain obfuscated code (Sucuri, Wordfence, etc.), but TARDIS can handle such well-known benign obfuscation cases by checking if it belongs to a CMS security plugin with licensing information.

V. Deploying TARDIS in the Wild

After validating that TARDIS's analysis accurately captures the attack labels in CMS-based website backups, we worked with CodeGuard to deploy TARDIS on a significant portion of their data set. We leveraged this access to nightly backups from 306,830 unique websites (spanning from March 2014 to May 2019) to empirically measure the health of CMS-based websites in the real world. In this section, we document our findings from using TARDIS to understand the threat landscape with respect to CMS-based websites. We are also in the process of working with CodeGaurd to inform the website owners of our findings and remediate the identified attacks.

Experimental Setup. We used a fleet of Amazon Web Services (AWS) Elastic Compute (EC2) r5.2xlarge instances with 8 virtual CPUs and 64 GB of RAM. These instances are supervised by the AWS Batch job scheduling engine to run TARDIS on hundreds of website backups in parallel.

We used several tools to assist in the investigation: Our CMS classification is built on top of WhatCMS [30] and CMS Garden [31]. TARDIS is written in Python (2500 lines of code) and leverages zxcvbn [27] for entropy

TABLE VI: Overall Distribution of Compromised Websites and Average File Counts per CMS.

CMS	Number of Websites	# Comp. Websites	Total Avg. Files Count	Only Comp. CMS	Only Benign CMS
WordPress	295,774	19,260	10,981.9	19,072.7	10,418.4
Drupal	1,340	215	17,760.0	22,288.7	16,894.5
Joomla	4,115	563	20,950.0	32,391.5	19,136.5
PivotX	509	27	28,739.7	42,075.9	27992.7
Prestashop	464	86	28,665.3	43,032.4	25,396.6
TYPO3 CMS	81	4	31,044.5	71,984.0	28,917.8
Contenido	4,543	436	16,709.8	25,851.5	15,739.3
Contao	4	0	7,634.0	NA	7,634.0

estimation in the injected element names and Pandas [32] for data analysis.

A. The CMS Landscape

Table VI presents the distribution of compromises in the 306,830 websites. Columns 1 and 2 show the CMS platform and its distribution, respectively. Column 3 shows the number of websites marked as compromised by TARDIS, i.e., the websites for which TARDIS outputs multi-stage attack labels and a compromise window. Columns 4 through 6 show the total average number of files ("spatial elements") for each CMS, the average number of files in compromised websites, and the average number of files in only the benign websites, respectively.

Table VI provides interesting insights into the attack landscape of CMSs. As seen in Column 2, the majority of the websites use WordPress as their underlying CMS. In this dataset, we see that 96% of the total websites use WordPress, higher than real-world trends [2]. This is due to the high market share of WordPress users in CodeGuard's production set. From Column 2 in Table VI, it is evident that, except for Contao, all CMSs in this dataset are victims of multi-stage attacks. In total, we found 20,591 compromised websites. There were 19,260 WordPress websites alone infected with these attacks (6.5% of the total WordPress websites). Interestingly, more than 16% of Joomla, 13% of Drupal, 18% of Prestashop, and 9% of Contenido websites were victims to multi-stage attacks. This goes to show that not only do these attacks target CMSs, but they target popular and the less popular CMSs alike. In this dataset, about 5% of PivotX and TYPO CMS3 websites were compromised by long-lived multi-stage attacks, showing that these CMSs might not be popular attack targets due to their smaller market share.

As seen in Column 4 from Table VI, almost all CMSs contain tens of thousands of files on an average. However, an interesting metric is to compare the average number of files in compromised CMSs with those in benign CMSs. Upon comparing Columns 5 and 6, it becomes evident that invariably the attacks inject an extremely large number of files into the CMS (which we also observed during our manual investigation). As highlighted in Table VI, almost all the compromised websites see a 50% or more increase in files. The highest bloat in the number of files is seen for TYPO3 CMS with a 150% increase in the average number of files upon compromise. WordPress stands second, which sees an average increase of 80%.

B. Evolution of Attacks

Table VII presents the distribution of attack models in the 20,591 websites that TARDIS identified as compromised. Rows 1 through 8 present these outputs for all the attack labels assigned by TARDIS. Recall that the assignment of these labels is described in Table III. Columns 2 through 8 show the number of websites marked as compromised by TARDIS for each CMS. A reading from Row 4 of Table VII can be interpreted as follows: After running TARDIS on a total of 295,774 WordPress websites in our dataset, it found 13,317 compromised websites with code generation capability. Lastly, Row 9 in Table VII presents the total number of compromises from each CMS for comparison.

From Table VII, it is evident that the code generation capability is the most common tactic, seen in more 70% of all attacks, regardless of the underlying CMS. From Row 1 of this table, we see that it is not common to identify the establish foothold label in all CMSs, mainly due to the nature of our dataset. However, when identified, it is a robust metric that confirms a multi-stage attack. It is also interesting to note that more than 20% of all such attacks attempt to clean up their traces after accomplishing the attack motive. However, not all multi-stage attacks actively hide their presence. As seen from Row 5 of Table VII, more than 60% of compromised WordPress websites try to evade defenses by following the popular hidden file/directory or the disguised file approaches. Conversely, the popular defense evasion techniques are not widely seen in compromised websites belonging to other CMSs. This could be attributed to the less-technical nature of the website owner due to which the adversaries do not spend resources on active hiding during the attack.

A significant portion of these attacks (8%) use obfuscation techniques to make it harder for the website owners to reverse engineer the injected code. Since the hosted websites cannot be taken down immediately upon detecting any traces of suspicious activity, by the time

TABLE VII: Attack Phase Distribution Across the 306,830 Websites.

Phases	WordPress	Drupal	Joomla	PivotX	Prestashop	TYPO3 CMS	Contenido	Contao
Establish Foothold	339	3	34	0	1	0	22	0
Obf. Code Injection	1,629	19	29	1	0	0	39	0
Malware Dropped	7,223	141	528	11	80	0	265	0
Code Gen. Capability	13,317	188	510	27	86	4	420	0
Defense Evasion	12,491	79	63	0	0	0	181	0
Escalate Privileges	12,078	72	55	17	6	4	176	0
Maintain Presence	1,704	27	45	1	0	0	37	0
Attack Cleanup	3,795	53	123	6	6	0	84	0
Total Number of Compromised Websites	19,260	215	563	27	86	4	436	0

incident response can understand the obfuscated code, the adversary would have completed reconnaissance in the websites (as presented in Figure 4), achieved their goals, and moved towards attack cleanup.

C. Compromise Window

The most important finding from this dataset was the length of compromise in CMS-based websites. Once attacked, multi-stage attacks persist in the websites for long periods of time. Figure 4 shows the compromise window distribution from the TARDIS output for each of the compromised CMSs. Note that this is a truncated version of the box plot to improve readability. Not shown in this figure: More than 20% of attacks on WordPress websites persist between 300 to 1694 days. As seen from Figure 4, most attacks in WordPress websites persist for around 40 days, as is evident from the median of the box-plot for WordPress. In comparison, the median length of the attacks is longer in Joomla and PivotX in the range of 75 to 85 days. In more than 4000 WordPress websites, these attacks persist anywhere between 3 months and 4.5 years. It is the multi-stage attacks belonging to this quartile (top 25%) that pose the most significant threat to website visitors: The dropped files simply lurk in the websites, many of which aim to exploit website visitors.

Among all CMSs, an average multi-stage attack persists the longest in Joomla, for 3 months on an average. Further, for websites that use more popular

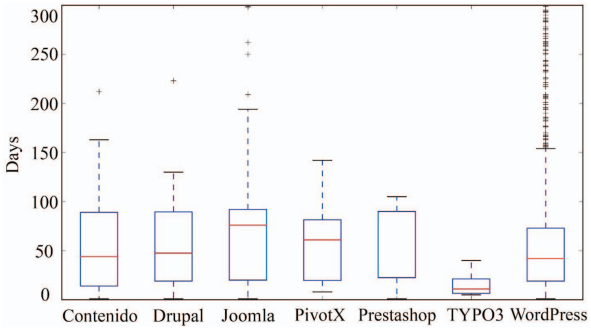

Fig. 4: Compromise window distribution in CMSs (truncated to 300 days).

TABLE VIII: Effectiveness of the Current Industry Attack Mitigation Framework.

CMS	Infected Websites[1]	AV Alerts[2]	Rollbacks[3]	Reinfects[4]
WordPress	19,260	52	17	7
Drupal	215	9	4	4
Joomla	563	28	7	7
PivotX	27	0	0	0
Pretashop	86	0	0	0
TYPO3	4	0	0	0
Contenido	436	2	1	1
Contao	0	0	0	0
Total	20,591	91	29	19

1: # of websites compromised for each CMS, 2: # of websites with AV alerts, 3: # of websites with attempted rollbacks, 4: # of websites with reinfections after the rollbacks.

CMSs such as WordPress, Joomla, and Drupal, the attacks likely persist longer since the adversaries see a wider opportunity base and get a better return on the investment of their resources. Conversely, the less popular CMSs, TYPO3 CMS and Prestashop, are not only targeted less by persistent attacks, but those attacks also do not persist for as long. This can be attributed to the higher opportunity cost and lower returns for targeting an attack toward less popular CMSs.

D. Existing Attack Mitigation Framework

Recall, the current industry standard is a naive "backup and restore" model in conjunction with an integrated AV. Once a compromise is detected by the AV, the website owners are prompted to rollback to a previous clean snapshot. We extracted the AV reports for the dataset of 306,830 websites and instrumented TARDIS to record the number of user-initiated rollbacks and reinfections post rollback. The results are presented in Table VIII. Note that TARDIS has no knowledge of the AV reports and relies only on its attack models for timeline reconstruction. Table VIII shows that AVs are ineffective in identifying almost all infected websites thus reaffirming our claim that AV signatures only catch well-known malware, and they fail to detect stealthy multi-stage attacks.

Columns 1 and 2 present the CMS platforms and the number of compromised websites from each CMS.

Column 3 shows the number of infected websites from each CMS that triggered AV alerts. Column 4 presents the number of websites with AV alerts that attempted to rollback to a previous version in order to mitigate the threat identified by the AV. Column 5 presents the number of websites that attempted rollbacks and remained infected or were reinfected. As expected, the distribution of the AV alerts and the rollbacks reflect the market share of the CMSs in CodeGuard's production set, which we consider representative of CMSs at large.

Rollbacks. As presented in Table VIII, we find it extremely concerning that among the 20,591 websites identified as compromised with long-lived multi-stage attacks by TARDIS, *only* 91 websites see AV alerts. More so, because the website owners are alerted to rollback to a clean snapshot only when an AV alerts the website owner about a suspicious activity. This low number of AV alerts (i.e. less than 1% of the total number of compromises) is the reason why the "backup and restore" model is proving to be ineffective and these attacks persist for a significant time period.

Among the 91 websites that trigger AV alerts, not all of them take action. As seen in Table VIII, only 29 of these 91 websites attempt a rollback to a pre-AV-alert snapshot to recover the website from the attack. Moreover, AVs are infamous for generating false alerts causing threat alert fatigue [24] — another reason why true AV alerts are ignored, perhaps explaining why only 29 of the 91 websites attempted rollbacks.

Reinfections. As seen from Table VIII, of the 29 websites that attempted rollbacks to recover from an attack, TARDIS found reinfections in more than 65% of these websites. We imagine this was quite confusing *to the attacker* to find the website files rolled back but their original backdoors *persisted*. This confirms the long-held belief that AVs are unreliable. Not only are they missing a vast majority of the attacks, but a strong dependence on AVs is making the existing "backup and restore" technique largely ineffective. These numbers reaffirm the motivation behind TARDIS's design — the need for a systematic provenance inference technique in the space of nightly backups. We hand-verified the websites with rollbacks and found that our intuition was correct: In every case of reinfection, the rollback snapshot was inside the compromise window (identified by TARDIS) causing a reinfection. Of all the compromised websites, only 10 websites managed to rollback outside of the compromise window, thus remediating the infections. This confirms that TARDIS's provenance inference is essential for compromised website investigation.

E. Performance

Figure 5 shows the time taken by TARDIS to measure all the attributes for 306,830 websites versus the size of the websites in terms of the number of files. TARDIS linearly assesses each temporal snapshot to provide a timeline of events and event labels for the entire website with acceptable overhead. While this overhead scales with the number of files in the website (regardless of the size of these files), the increase is minimal as is seen from the gradual slope of the plot in Figure 5. The worst-case for TARDIS, i.e., the maximum time taken, was to process 1859 snapshots (an average of 100,000 spatial elements) is close to 3500s. As an offline forensics technique, we consider this to be quite reasonable.

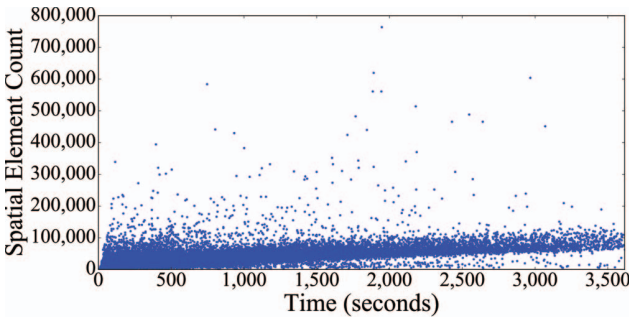

Fig. 5: Time to process a CMS backup (seconds) versus total number of files in the CMS.

VI. Case Study

A. Case Study 1: A Global View of Attack Movement

Beyond the investigation of individual websites, deploying TARDIS *within* commercial website backup platforms [19]–[23] can provide a global view of the evolution of attack campaigns. During our experiments, we found identical provenance evolution across 5 different WordPress websites between September and November 2018. In all of these websites, the adversary uses similar tactics of disguised *obfuscated code injection* (O) in 28 PHP files in different locations over 5 days, followed by 83 instances of *malware dropped* (M) to inject backdoor functionalities, then *maintaining presence* (P) for about 2 months, and eventually attempting *attack cleanup* (C) to remove all traces of those steps. In each of these cases, the dropped malware disables all error logging functionalities and fetches payloads from a remote server (active at the time of investigation) which it executes on the victim web server. It also collects the output buffers, sends them back to the remote server, and finally re-enables error logging. These actions were programmed to run every 48 hours.

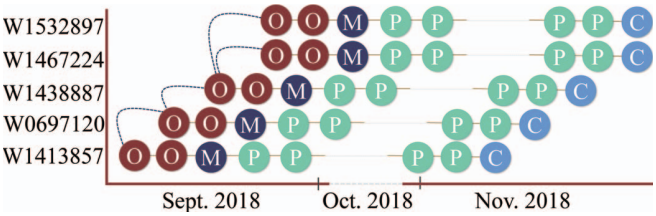

Fig. 6: Global attack movement in WordPress websites.

Interestingly, as shown in Figure 6, each of these websites exhibited the same attack phase evolution and

persistence for the same duration. All of these websites belong to different unique small businesses that just happen to build their website upon WordPress. Once the first WordPress website (`W1413857`) was attacked, another WordPress website (`W0697120`), completely unrelated to the initial website, exhibits the exact same injection 6 days later. This is followed by 3 other infections in three websites (`W1438887, W1467224, W1532897`) within the next 10 days. In all five of these websites, the obfuscated code injection phase lasts for 5 days, malware dropped phase for 1 day, maintain presence 51 to 56 days, and finally, the attack is cleaned up by deleting all the injected files. Since this attack was not known to the AV, none of the attacks were flagged and the website owners did not attempt rollback.

Future Deployment. We are currently working with CodeGuard to deploy TARDIS at a global level in their backup framework to detect and track large scale attack trends. This has required expanding TARDIS to enable cross-website modeling and correlation.

B. Case Study 2: "User-Friendly" Remote Control

In a Drupal-based website, investigation of the backups for a 3 month period (Feb 2019 - Apr 2019) revealed the existence of the following phases.

```
23 Feb           Obfuscated Code Injection
24 Feb - 3 Mar   Maintain Presence
4 Mar            Malware Dropped & Defense evasion
6 Mar            Escalate Privileges
7 Mar - 12 Apr   Maintain Presence
13 Apr           Attack Cleanup
```

The integrated AV at the backup site never triggered an alert, keeping the website owner in the dark about the attack. The compromise window identified by TARDIS showed that the adversary injected obfuscated PHP code starting 23 February 2019 and maintained presence for the next few days until 3 March 2019. Starting 4 March 2019, the attacker dropped malware and used defense evasion methods: They disguised a PHP file as an icon file and uploaded a backdoor shell inside a hidden directory. On March 6, the attackers injected a full-fledged graphical user interface (GUI) for the backdoor, giving them full control of the website as shown in Figure 7. All of these files remained in the website for over a month. On 13 April 2019, TARDIS found that the attackers deleted the earlier injected malware to hide their previous footprints. The timeline provided by TARDIS reveals that multi-stage attack activities persisted in this website during a period of 3 months and provides a compromise window (23 February to 13 April 2019) outside of which the website can be safely rolled back. Manual investigation revealed that the CVE-2018-7600 [33] vulnerability, insufficient input sanitation on Form API (FAPI) AJAX requests, was exploited by the attacker. Note that this vulnerability was not patched until a month after this attack began.

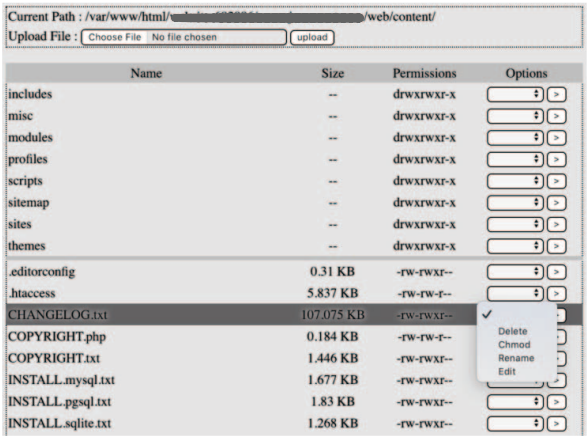

Fig. 7: GUI backdoor injected in a Drupal website.

VII. Limitations

The accuracy of inferring the provenance of attacks is limited by the granularity of the backups. The current industry norm is to collect website backups nightly [19]–[23]. We have shown that this is sufficient for recovering the timeline of an attack. However, if a fast-paced attack goes through multiple stages in between two consecutive backups, TARDIS would only have visibility into files at the time of the backups. Essentially, TARDIS enables website owners to calibrate between the granularity of taking backups (i.e., making TARDIS more accurate) versus requiring a deeper manual inspection when an attack does occur.

As these multi-stage attacks evolve, TARDIS's spatial metric identification rules might need to change over time. This evolution is expected, and TARDIS's modular nature makes adding new spatial metrics straightforward. Further, TARDIS's methodology of temporal correlation of spatial metrics should stand the test of time, as it was designed agnostic to the individual metrics and is based on the invariants of the phases which multi-stage attacks go through.

VIII. Related Work

Large-Scale Study of Web Attacks. Several studies have been published which used high-interaction honeypots to understand web attacks on a large scale [26], [34], [35]. Further, some techniques tried to assess the impact of web application compromises by studying the role of hosting providers [36] and understanding the response landscape by studying large-scale notification campaigns [37]. Similar to TARDIS, Canali et.al. [26] also found attackers dropping large volumes of files on the web server. While these techniques focused on attacks targeting a generic web application, this research studied the spread of multi-stage attacks on CMS-based websites and, more specifically, within production websites. Our study and TARDIS were designed to investigate such multi-stage attacks based on only nightly website backups.

Causality Modeling. There have been significant advances in identifying the provenance of an attack by monitoring the behavior of a system in order to reconstruct the chain of events that led to the attack [4], [7], [12], [13], [17], [38], [39]. Most works which focused on advanced multi-stage attack detection, e.g., HOLMES [3] and SLEUTH [16], are built on OS audit data and used system-call level logs for real-time analytics. However, these fine-grained log-based provenance tracking techniques require significant instrumentation and are hardly deployed by CMS hosting companies. TARDIS leverages what is already the industry standard (nightly backups) to model long-lived multi-stage attack progression via temporal correlation of spatial metrics and outlier detection.

Web Application Security. To preemptively secure websites against attack, recent research has focused on analyzing particular classes of attacks, such as ad injection [40]–[42], survey scams [43], [44], cross-site scripting [45]–[48], PHP code injection [49], [50], SQL injection [48], [51]–[53], file inclusion attacks [54], [55], etc. These research techniques focus on individual layers of web applications. However, since CMSs contain code across all of these layers and are marketed to less-technical website operators, attack-vector-specific solutions are not commonly deployed. TARDIS is attack-vector agnostic and enables the investigation of a compromised CMS post-attack.

IX. Conclusion

This paper presented a systematic study of the CMS attack landscape across 306,830 unique production websites, using TARDIS. Targeting the problem of investigating compromises in CMS-based websites using *only* the readily available nightly backups, TARDIS provides a novel provenance inference technique that reconstructs the attack phases and enables rapid recovery from an attack. Using the temporal correlation of spatial metrics representing each snapshot, TARDIS recovers the compromise window and the progression of attack phases. TARDIS uncovered 20,591 websites that were victims of long-lived multi-stage attacks and was shown to be highly accurate in revealing attacks in CMS-based websites, regardless of the underlying CMS. We are working with CodeGuard to informing the website owners and remediate the identified attacks.

X. Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and feedback. We also thank Professor Manuel Egele for his guidance while shepherding this paper, and our collaborators at CodeGuard for their insightful comments and suggestions throughout this research. This work was supported, in part, by NSF under Award 1916550. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors or collaborators.

References

[1] *W3Techs - Usage of content management systems for websites*, https://w3techs.com/technologies/overview/content_management/all, [Accessed: 2019-01-16].

[2] *Popular CMS by Market Share*, https://websitesetup.org/popular-cms/, [Accessed: 2019-06-30].

[3] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-time apt detection through correlation of suspicious information flows," in *Proc. 40th IEEE S&P*, San Francisco, CA, May 2019.

[4] 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 *Proc. 26th USENIX Sec.*, Vancouver, BC, Canada, Aug. 2017.

[5] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "MCI: Modeling-based causality inference in audit logging for attack investigation," in *Proc. 2018 NDSS*, San Diego, CA, Feb. 2018.

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

[7] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition," in *Proc. 20th NDSS*, San Diego, CA, Feb. 2013.

[8] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage collecting audit log," in *Proc. 20th ACM CCS*, Berlin, Germany, Oct. 2013.

[9] 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 *Proc. 31st ACSAC*, 2015.

[10] 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 *Proc. 2017 NDSS*, San Diego, CA, Feb. 2017.

[11] 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 *Proc. 2018 USENIX Annual Technical Conference (ATC)*, Boston, MA, Jul. 2018.

[12] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards practical provenance tracing by alternating between logging and tainting," in *Proc. 2016 NDSS*, San Diego, CA, Feb. 2016.

[13] S. Sitaraman and S. Venkatesan, "Forensic analysis of file system intrusions using improved backtracking," in *Proc. 3rd IEEE International Workshop on Information Assurance*, IEEE, College Park, MD, USA, Mar. 2005.

[14] 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 *Proc. 32nd ACSAC*, 2016.

[15] F. Wang, Y. Kwon, S. Ma, X. Zhang, and D. Xu, "Lprov: Practical library-aware provenance tracing," in *Proc. 34th ACSAC*, 2018.

[16] 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 *Proc. 26th USENIX Sec.*, Vancouver, BC, Canada, Aug. 2017.

[17] 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 *Proc. 2018 NDSS*, San Diego, CA, Feb. 2018.

[18] *HostGator.com LLC*, https://www.hostgator.com, [Accessed: 2019-06-12].

[19] *Dropmysite - Cloud Backups for Websites & Databases*, https://www.dropmysite.com/, [Accessed: 2018-10-31].

[20] *CodeGuard*, https://www.codeguard.com/, [Accessed: 2019-01-20].

[21] *GoDaddy*, https://www.godaddy.com/web-security/website-backup, [Accessed: 2018-01-20].

[22] *Sucuri*, https://sucuri.net/website-backups/, [Accessed: 2018-10-31].

[23] *iPage*, https://www.ipage.com/web-backup, [Accessed: 2018-10-31].

[24] 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 *Proc. 2019 NDSS*, San Diego, CA, Feb. 2018.

[25] *New Research from Advanced Threat Analytics*, https://www.prnewswire.com/news-releases/new-research-from-advanced-threat-analytics-finds-mssp-incident-responders-overwhelmed-by-false-positive-security-alerts-300596828.html, [Accessed: 2019-01-20].

[26] D. Canali and D. Balzarotti, "Behind the scenes of online attacks: An analysis of exploitation behaviors on the web," in *Proc. 20th NDSS*, San Diego, CA, Feb. 2013.

[27] *zxcvbn: Low-Budget Password Strength Estimation*, https://github.com/dropbox/zxcvbn, [Accessed: 2019-05-28].

[28] D. C. Howell, "Median absolute deviation," *Wiley StatsRef: statistics reference online*, 2014.

[29] A. Koufakou, E. G. Ortiz, M. Georgiopoulos, G. C. Anagnostopoulos, and K. M. Reynolds, "A scalable and efficient outlier detection strategy for categorical data," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, IEEE, vol. 2, 2007, pp. 210–217.

[30] *What CMS Is This Site Using?* https://whatcms.org/, [Accessed: 2019-06-26].

[31] *CMS-Garden CMSScanner*, https://github.com/CMS-Garden/cmsscanner, [Accessed: 2019-06-12].

[32] *Pandas: Flexible and powerful data analysis and manipulation library for Python*, https://github.com/pandas-dev/pandas, [Accessed: 2019-05-28].

[33] *Drupal: CVE-2018-7600: Remote Code Execution - SA-CORE-2018-002*, https://www.rapid7.com/db/vulnerabilities/drupal-cve-2018-7600, [Accessed: 2019-06-26].

[34] O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N. Nikiforakis, "No honor among thieves: A large-scale analysis of malicious web shells," in *Proc. 25th WWW*, 2016.

[35] O. Catakoglu, M. Balduzzi, and D. Balzarotti, "Automatic extraction of indicators of compromise for web applications," in *Proc. 25th WWW*, 2016.

[36] D. Canali, D. Balzarotti, and A. Francillon, "The role of web hosting providers in detecting compromised websites," in *Proc. 22nd WWW*, Rio de Janeiro, Brazil, May 2013.

[37] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes, "Hey, you have a problem: On the feasibility of large-scale web vulnerability notification," in *Proc. 25th USENIX Sec.*, Austin, TX, Aug. 2016.

[38] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 2003.

[39] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. L. MacLean, D. W. Margo, M. I. Seltzer, and R. Smogor, "Layering in provenance systems.," in *Proc. 2009 USENIX Annual Technical Conference (ATC)*, San Diego, CA, Jun. 2009.

[40] S. Arshad, A. Kharraz, and W. Robertson, "Identifying extension-based ad injection via fine-grained web content provenance," in *Proc. 19th RAID*, Evry, France, Sep. 2016.

[41] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. Abu Rajab, "Ad injection at scale: Assessing deceptive advertisement modifications," in *Proc. 36th IEEE S&P*, San Jose, CA, May 2015.

[42] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee, "Understanding malvertising through ad-injecting browser extensions," in *Proc. 24th WWW*, 2015.

[43] A. Kharraz, W. Robertson, and E. Kirda, "Surveylance: Automatically detecting online survey scams," in *Proc. 39th IEEE S&P*, San Francisco, CA, May 2018.

[44] J. W. Clark and D. McCoy, "There are no free ipads: An analysis of survey scams as a business.," in *Proc. 6th USENIX LEET*, Washington, D.C., United States, Aug. 2013.

[45] W. Melicher, A. Das, M. Sharif, L. Bauer, and L. Jia, "Riding out domsday: Toward detecting and preventing dom cross-site scripting," in *Proc. 2018 NDSS*, San Diego, CA, Feb. 2018.

[46] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns, "Precise client-side protection against dom-based cross-site scripting.," in *Proc. 2014 NDSS*, San Diego, CA, Feb. 2014.

[47] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," in *Proc. 30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, May 2008.

[48] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi, "Efficient and flexible discovery of php application vulnerabilities," in *Proc. European Symposium on Security and Privacy (EuroS&P)*, IEEE, Paris, France, Apr. 2017.

[49] D. R. Sahu and D. S. Tomar, "DNS pharming through PHP injection: Attack scenario and investigation," *IJ Computer Network and Information Security*, vol. 4, pp. 21–28, 2015.

[50] V. Yerram and G. V. R. Reddy, "A solution to php code injection attacks and web vulnerabilities," 2014.

[51] Z. S. Alwan and M. F. Younis, "Detection and prevention of sql injection attack: A survey," *International Journal of Computer Science and Mobile Computing*, vol. 6, no. 8, pp. 5–17, 2017.

[52] N. Singh, M. Dayal, R. Raw, and S. Kumar, "Sql injection: Types, methodology, attack queries and prevention," in *Proc. 3rdComputing for Sustainable Global Development (INDIACom)*, IEEE, New Delhi, India, Mar. 2016.

[53] A. Pramod, A. Ghosh, A. Mohan, M. Shrivastava, and R. Shettar, "Sqli detection system for a safer web

application," in *Proc. 2015IEEE International Advance Computing Conference*, IEEE, Bangalore, India, Jun. 2015.
[54] H. F. G. Robledo, "Types of hosts on a remote file inclusion (rfi) botnet," in *Proc. Electronics, Robotics and Automotive Mechanics Conference*, Jun. 2008.
[55] O. Katz, "Detecting remote file inclusion attacks," *White Paper. Breach Security Inc., May*, 2009.