# A Simple and Novel Technique for Counteracting Exploit Kits

Byungho Min(✉) and Vijay Varadharajan

Advanced Cyber Security Research Centre, Department of Computing,
Macquarie University, Sydney, Australia
{byungho.min,vijay.varadharajan}@mq.edu.au

**Abstract.** Exploit kits have become a major cyber threat over the last few years. They are widely used in both massive and highly targeted cyber attack operations. The exploit kits make use of multiple exploits for major web browsers like Internet Explorer and popular browser plug-ins such as Adobe Flash and Reader. In this paper, a proactive approach to preventing this prevalent cyber threat from triggering their exploits is proposed. The suggested new technique called `AFFAF` proactively protects vulnerable systems using a fundamental characteristic of the exploit kits. Specifically, it utilises *version information* of web browsers and browser plugins. `AFFAF` is a zero-configuration solution, which means that users do not need to configure anything after installing it. In addition, it is an easy-to-employ methodology from the perspective of plugin developers. We have implemented a lightweight prototype and have shown that `AFFAF` enabled vulnerable systems can counteract 50 real-world and one locally deployed exploit kit URLs. Tested exploit kits include popular and well-maintained ones such as Blackhole 2.0, Redkit, Sakura, Cool and Bleeding Life 2. We have also demonstrated that the false positive rate of `AFFAF` is virtually zero, and it is robust enough to be effective against real web browser plugin scanners.

**Keywords:** Exploit kit · Malware · Web browser security

## 1 Introduction

In recent years, attacks targeting web browsers and browser plugins have become one of the most prevalent threats [1,2]. These attacks exploit vulnerabilities in the web browsers, their plugins and operating systems in order to download and execute malicious software on the victim system. This kind of attack is called "drive-by download", and attacks known as "exploit kits" (or exploit pack). An exploit kit contains several exploits that can compromise diverse systems from old Windows XP to recent Windows 7. Typically the range of the exploits included in a single exploit kit usually covers all the popular web browsers and plugins such as Flash, Adobe Reader and Java so as to maximise the possibility of successful compromise [3–5]. Also, exploit kits are used in various cyber attacks from massive spamming to highly sophisticated APT like Aurora operation [6].


J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 259–277, 2015.
DOI: 10.1007/978-3-319-23829-6_19

As the number of drive-by download attacks and that of exploit kits increase, several techniques to detect or prevent them have been proposed [6–12]. These techniques use one or more static and dynamic features such as characteristics and behaviours of malicious web pages. Another approach in the industry is giving users an option to block (or allow) web browser plugins entirely or selectively based on blacklisted (or whitelisted) web sites [13]. Major browsers like the Internet Explorer, Chrome, Firefox and Safari have basic features for enabling or disabling plugins, while a few web browser plugins like ClickToPlugin[1] and FlashBlock[2] provide more controls over the plugins such as whitelist. In some cases, operating system blocks outdated plugins [14].

In this paper, we propose a new approach to the exploit kit problem. *Rather than reactively detecting and blocking exploit kits, our approach proactively modifies certain behaviour of the web browser plugins in order to prevent exploit kits from triggering their exploits.* We have analysed multiple exploit kits and discovered a fundamental difference between benign software developers and malicious exploit kit developers; they both detect and check the version of to-be used plugin, but the way they check it is completely opposite to each other; we describe this difference in approach in Sect. 3. This observation led us to the proposed defensive methodology, `AFFAF (A Fake for a Fake)`, which leverages the difference to limit exploit kits' activities, while allowing normal web sites to function as intended. From a security perspective, `AFFAF` has several advantages. `AFFAF` is more fine grained than the allow/block-based solutions in the sense that it uses *version information* of the browser plugins. Next, it protects vulnerable systems as well as fully updated ones by thwarting exploit kits at the very early stage of an attack. In addition, it is hard for attackers to bypass `AFFAF` even after they know this methodology, since it makes use of an essential feature of exploit kits. Furthermore, `AFFAF` is a zero-configuration solution, hence users do not need to make decisions on whether to enable a specific browser plugin or not every time they visit a new web site. Finally, `AFFAF` is easy to apply and adopt as a practical technique.

We have implemented a prototype based on our methodology. Our prototype implementation uses JavaScript and web browser extension techniques to intercept communications between the web browser and web pages, and to modify the behaviour of web browser plugins. Because checking browser plugin versions using JavaScript is the *de-facto* standard among web developers and attackers, our prototype successfully blocks multiple real-world exploit kits including Blackhole 2.0, Redkit, Sakura, Cool and Bleeding Life 2, which proves the efficacy of the proposed methodology. Our evaluation involved Alexa top 100 web sites with Flash and/or PDF contents as well as ten fully Flash-based sites. In all theses cases, deployment of our prototype worked very well, suggesting the false positive rate of zero in these cases.

[1] http://hoyois.github.io/safariextensions/clicktoplugin/.
[2] https://chrome.google.com/webstore/detail/flashblock/gofhjkjmkpinhpoiabjplobcaignabnl.

The remainder of this paper is organised as follows. In Sect. 2, an overview on exploit kits is given. Our defensive methodology, `AFFAF`, is explained in Sect. 3 along with our observations on exploit kits. Prototype implementation is described in Sect. 4. Three types of evaluation and their results are described in Sect. 5. We discuss related work in Sect. 6, and then conclude this paper with some final remarks in Sect. 7.

## 2 Background

Discussion on exploit kits and detection techniques is given in this section providing the background knowledge for the problems addressed in this paper. We also present our observations on the exploitation strategy being used in many exploit kits.

*Implementation of Exploit Kits:* Exploit kits consist of two parts: malicious pages (client-side) and a control panel (server-side). When a victim visits a malicious link contained in a web site or a spam whose server has been compromised and poisoned with malicious contents, the user is redirected to the landing page of the exploit kit (after passing through several intermediary servers). Then the exploit kit profiles the victim environment using client-side script such as JavaScript. Based on the information gathered, it determines, delivers and launches one or more exploits amongst many available exploits for web browsers and their plugins. If the exploitation is successful, a malware is downloaded and executed on the victim system. This process is called "drive-by download" attack because it happens without any user consent [3]. The malware starts communicating with its control panel/administration interface that provides control functionalities such as remote access and various statistics on the compromised systems. Malicious pages delivered to victims are constructed using HTML and JavaScript, while control panel used by attackers is written using server-side scripting languages like PHP and backend software such as MySQL and Apache.

*Defensive Techniques:* As exploit kits have become a major cyber threat [1,2], several techniques to detect them have been proposed. One of the initial efforts was to detect malicious domains/URLs [8,10,15–18] and to build blacklists of these domains using techniques such as Google Safe Browsing, Malware Domain List and URL Query. To counteract the efficacy of such defence, attackers began to use fast-flux DNS and obfuscated code. Fast-flux DNS makes blacklists hard to be kept up-to-date, and obfuscated and evasive code enables the attackers (1) to avoid signature or other static feature-based detection techniques and (2) to protect the code from being analysed [19]. As a consequence, dynamic feature-based and behaviour-based detection techniques have been proposed in [6,7], and the arms race continues. These trends in attacks and defence mechanisms are exactly same as malware history that began with normal binaries and then evolved into obfuscation bypassing signature-based anti-virus.

**Table 1.** CVEs exploited in the tested exploit kits and corresponding vulnerable products with version information

| CVE | Vulnerable versions | CVE | Vulnerable versions |
|---|---|---|---|
| CVE-2007-5659/ 2008-0655 | Adobe Reader ≤ 8.1.1 | CVE-2011-1255 | IE 6-8 |
| CVE-2009-2477 | Firefox ≤ 3.5.1 | CVE-2011-2110 | Flash ≤ 10.3.181.26 |
| CVE-2008-2992 | Adobe Reader ≤ 8.1.2 | CVE-2011-2140 | Flash ≤ 10.3.183.5 |
| CVE-2008-5353 | Java ≤ 6u10 | CVE-2011-2371 | Firefox ≤ 4.0.1 |
| CVE-2009-0927 | Adobe Reader ≤ 9.1 | CVE-2011-2462 | Adobe Reader ≤ 10.1.1 |
| CVE-2009-3867 | Java ≤ 6u17 | CVE-2011-3106 | Chrome ≤ 19.0.1084.52 |
| CVE-2009-4324 | Adobe Reader ≤ 9.3 | CVE-2011-3659 | Firefox ≤ 3.6.26, 4.9 |
| CVE-2010-0188 | Adobe Reader ≤ 9.3.1 | Firefox social engineering | Firefox ≤ 4 |
| CVE-2010-0094 | Java ≤ 6u18 | CVE-2012-0754 | Flash ≤ 10.3.183.15, 11.1.102.62 |
| CVE-2010-0840 | Java ≤ 6u18 | CVE-2012-0775 | Adobe Reader ≤ 10.1.3, ≤ 9.5.1 |
| CVE-2010-0842 | Java ≤ 6u18 | CVE-2012-0779 | Flash ≤ 10.3.183.19, 11.2.202.235 |
| CVE-2010-0886 | Java ≤ 6u19 | Unknown CVE | Flash ≤ 10.3.183.23, 11.4.402.265 |
| CVE-2010-1240 | Adobe Reader ≤ 9.3.3 | CVE-2012-3683 | Safari ≤ 6 |
| CVE-2010-1297 | Flash ≤ 10.1.53.64 | CVE-2012-4681 | IE 6-9 |
| CVE-2010-1885 | Windows XP-2003 | CVE-2012-4792 | IE 6-8 |
| CVE-2010-0248 | IE 6-8 | CVE-2012-4969 | IE 6-9 |
| CVE-2010-2883 | Adobe Reader ≤ 9.4 | CVE-2012-5076 | Java ≤ 7u8 |
| CVE-2010-2884 | Flash ≤ 10.1.82.76 | CVE-2012-1880 | IE 6-9 |
| CVE-2010-3552 | Java ≤ 6u21 | CVE-2012-1876 | IE 6-10 |
| CVE-2010-3654 | Flash ≤ 10.1.102.64 | CVE-2012-1889 | Windows XP-7, Server 2003-2008 |
| CVE-2010-4452 | Java ≤ 6u23 | CVE-2013-0422 | Java ≤ 7u11 |
| CVE-2011-0558 | Flash ≤ 10.2.152.26 | CVE-2013-0634 | Flash ≤ 10.3.183.51 |
| CVE-2011-0559 | Flash ≤ 10.2.152.26 | CVE-2013-1493 | Java ≤ 7u15 - 6u41 |
| CVE-2011-0611 | Flash ≤ 10.2.154.27 | CVE-2013-2423 | Java ≤ 7u17 |

## 3 AFFAF: Proposed Methodology

In this section, we give our observations on the exploitation strategy of exploit kits', and then propose a novel defensive methodology to counteract it. As a proactive solution, our technique can be combined with any reactive defensive technique from blacklists to malicious page/code detection mentioned earlier.

*Observations on Exploitation Strategy of Exploit Kits:* In addition to evasion techniques, there is another important commonality that exploit kits share to install silently malware without user's notice; they profile the victim system before launching exploits. After analysing major exploit kits, such as Blackhole 2.0 and Bleeding Life 2, we observed that they all use similar exploit determination process like the one shown in Fig. 1. It is a flow chart representation of Blackhole 2.0's JavaScript code for its Flash exploits. Not only Flash, but also browser type and other conditions are tested in the diagram. This kind of version detection is confirmed in many exploit kits [2,3,5,20–24]. In a couple of ways, profiling is a crucial strategy of exploit kits that enables reliable and secret compromise. First, it is well-known that unsuccessful software exploitation may make the target web browser or plugin be unresponsive or crash, which

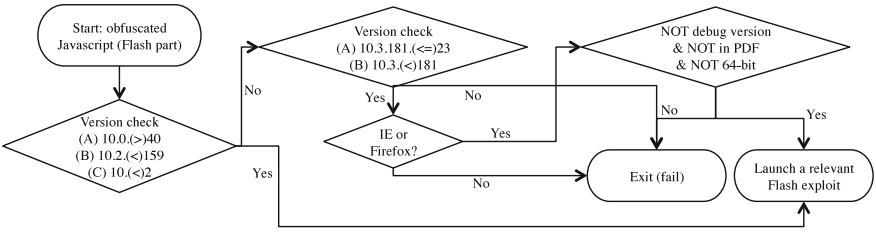


**Fig. 1.** Control flow for Flash exploitation (Blackhole 2.0)

will cause suspicions. We have experimented with several exploits and analysed exploit kits (Table 1), which confirmed this behaviour. In the worst case, such a crash can lead to the detection of an entire attack operation in which the exploit kit is involved in. Obviously, the attackers do not want this to happen. What the attackers usually want is to compromise as many victims as possible and remain undetected as long as possible. Therefore, attacking only vulnerable targets and avoiding highly secured (not vulnerable) targets is acceptable for them. As a result, they avoid non-vulnerable versions where their exploitation attempts will not succeed. This is why exploit kits exquisitely pinpoint the vulnerable web browsers and plugins as given in Fig. 1, prior to downloading and launching an exploit. Even penetration testing tools like Metasploit[3] and SET (Social Engineering Toolkit)[4] check the target version before trying exploits in order to reliably and secretly compromise target machines. Second, exploit kits contain more than one exploit. In other words, it can try other exploits (e.g. Internet Explorer browser exploit) even if a vulnerable version of one plugin (e.g. old Java) is not installed on the target. This strategy raises the possibility of successful exploitation as well as reducing the risk of exposure. Third, by delivering only the required exploit, only a portion of a complete exploit kit is exposed to victims and security experts, thus making the analysis difficult.

*Developers vs. Attackers:* After observing the profiling behaviour of exploit kits, we discovered a fundamental difference between attackers and web developers. Even though they both use web browser plugins, the attackers exploit them in a way to compromise victim machines, whereas the developers utilise their functionalities in order to implement rich web applications. And before actually using a plugin, both the attackers and the developers check the existence and/or version of the plugin. However, the way they check the version number is totally different. Benign developers normally check the existence or the **minimum** version number required for their web applications (Java 1.5.XX or higher, for example); it is a commonly accepted development practice to require a specific or higher version of software. The developers perform this check for compatibility. If the required plugin is not installed, or its version is too low

[3] http://www.metasploit.com.
[4] http://www.social-engineer.org.

and some required functionalities are not provided in the old version, the web application cannot run on the system. In addition, thanks to backward compatibility practice that is universally deployed in the software industry, developers normally don't have to check the upper limit for compatibility. For instance, the following code from Adobe's Flash development help page[5] shows how a Flash content is embedded in HTML. It only checks the existence of Flash plugin, and displays "Get Adobe Flash player" link if Flash is not available on the client system:

```
<object classid="clsid:d27cdb6e..." \
        width="550" height="400" \
        id="movie_name" align="middle">
  <param name="movie" value="movie_name.swf" />
  <!--[if !IE]>-->
  <object type="application/x-shockwave-flash" \
          data="movie_name.swf" width="550" ...>
      <param name="movie" value="movie_name.swf" />
  <!--<![endif]-->
      <a href="http://www.adobe.com/go/getflash">
          <img src="http://www.adobe.com/images/ \
          .../get_flash_player.gif" \
          alt="Get Adobe Flash player" />
      </a>
  <!--[if !IE]>-->
  </object>
  <!--<![endif]-->
</object>
```

In contrast, the **maximum** version number required for successful exploitation (for instance, Java 7u11 or lower) is probed by the attackers. This *equal to or less than* tendency is evident in Table 1. This table shows CVEs exploited by the exploit kits that we have analysed and tested.[6] The reason for this tendency is clear. The attackers need to profile victim systems for the reasons discussed earlier, and the profiling code checks equal to or less than relation because a vulnerability is applied to a specific version or below, as discussed earlier. For instance, if a zero-day vulnerability is disclosed for Flash and version 10.2.158 was the latest at that time, the exploit code for the vulnerability is applicable to 10.2.158 or below. As a concrete example, a notorious exploit kit, Blackhole 2.0, checks an exact version range before trying its Flash exploits as given below (see Fig. 1 for more detail):

```
function spl5() {
  var ver1 = flashver[0];
  var ver2 = flashver[1];
  var ver3 = flashver[2];
  if (((ver1 == 10 && ver2 == 0 && ver3 > 40) \
     ||((ver1 == 10 && ver2 > 0) && \
       (ver1 == 10 && ver2 < 2))) \
     ||((ver1 == 10 && ver2 == 2 && ver3 < 159) \
     ||(ver1 == 10 && ver2 < 2))) {
      // Embed Flash Exploit(s)
}
```

[5] http://helpx.adobe.com/flash/kb/object-tag-syntax-flash-professional.html.

[6] In order to give correct information, all the data of this table is verified using the official CVE web site and ExploitPack Table 2013 that are available at http://cve.mitre.org and https://docs.google.com/spreadsheet/ccc?key=0AjvsQV3iSLa1dE9EVGhjeUhvQTNReko3c2xhTmphLUE respectively.

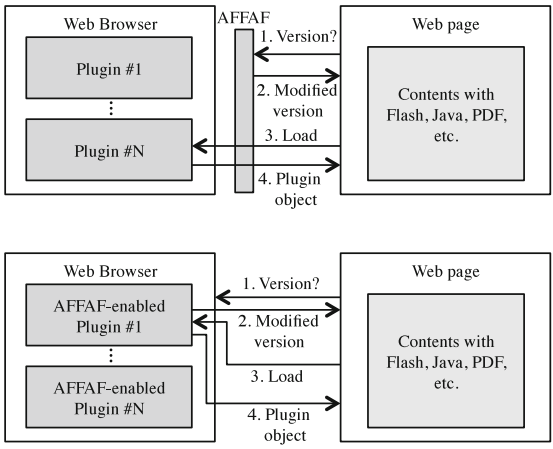


**Fig. 2.** Proposed mitigation against exploit kits for browser plugins: separate (upper) or integrated (lower)

The JavaScript code above meticulously pinpoints the target Flash versions such as 10.2.{0-158} and 10.{0 or 1}.{XXX}, and loads malicious Flash contents only if one of the conditions is satisfied. Then, action script inside the Flash contents executes the actual exploit code. Even though JavaScript is the most popular place where exploit kits perform their version detection (given in Sect. 5), a few exploit kits check target software version from their malicious payload. For instance, Redkit exploit kit conducts version detection inside its PDF payload.

*Leveraging the Difference:* The current situation that developers check the minimum required version, while attackers tend to check the maximum vulnerable version can be utilised to protect systems from exploit kits. In most cases, exploits included in exploit kits are not triggered if the version of the target browser plugin is higher than the maximum vulnerable version (which means patched). Therefore, by altering outdated plugins to be the latest (or even a non-existent future version), we can make all the conditions probed by exploit kits (such as those of Fig. 1) fail; hence helping to prevent the launch of exploits. Suppose that a plugin (e.g. Java) advertises its version number to be higher than its actual version (e.g. the latest or even higher one), it prevents exploit kits from trying Java exploits that target some old versions or the latest version in the case of zero-day. More importantly, users can still enjoy Java applets on benign web sites, since those web sites check either the existence of Java or minimum required version, which is definitely lower than the fake version. As a result, we can block plugin exploits contained in exploit kits, while leaving browser plugins usable to normal web applications. We call this defensive methodology `AFFAF` (a fake for a fake), meaning that it uses fake version numbers to thwart exploit kits and drive-by download attacks. With regard to false positive (the case when benign web application is blocked as well as exploit kits), it is expected to be low

because security fixes are normally minor version updates and no new features are introduced.

This methodology can be implemented as either a separate solution (this paper's prototype) or an integrated part of each plugin. Each case is depicted in Fig. 2. In the former case, `AFFAF` intercepts version enquires and returns a fake one, whereas in the latter case, each plugin is responsible for such manipulation. In both cases, a web browser plugin or `AFFAF` (1) responds with the requested plugin object to the web page, and (2) reports the real version information to relevant vendor so that update is possible, while reporting fake versions to any other web sites.

### 3.1 Merits of AFFAF

`AFFAF` is a fine-grained defence using version numbers of plugins that helps to counteract attacks. As a consequence, several benefits including the following major ones are obtained:

1. `AFFAF` is a zero-configuration solution. Users do not need to disable their plugins nor blacklist (or whitelist) them; in other words, users need to do nothing for `AFFAF`. This means simple convenience. The users can use old and vulnerable browser plugins without worrying about being exploited by exploit kits. This is crucial in web user protection, because many users (93 % of Java and 60 % of Adobe Reader users) do not update their plugins and use outdated (thus vulnerable) ones [25,26]. Even worse, less than one per cent of enterprises run the latest version of Java [27]. Some users even want to disable security warnings for outdated browser plugins provided by browsers [28]. Our methodology protects these old versions as well as the latest ones without disabling them.
2. Even after attackers know `AFFAF` methodology, it is still hard for them to bypass it. Attackers cannot try an arbitrary exploit, since they do not know the real versions of browser plugins. For instance, suppose an exploit kit has an exploit for CVE-2012-0779. It works against Flash up to 10.3.183.19 for version 10 and up to 11.2.202.235 for version 11 (Table 1). Even after the attackers guess Flash version obtained from the victim is a fake, they cannot try this exploit because it may crash the browser or make it unresponsive, which makes the user suspect an attack. As a result, the attackers hardly use exploits against seemingly not vulnerable environment as shown earlier. Until they find a way to obtain the real version number, promiscuously trying exploits is too risky for the attackers.
3. `AFFAF` thwarts attacks at the very early stage. Without using their exploits, exploit kits cannot use any further exploits such as Windows privilege escalation that are supposed to be triggered by the first web browser exploits, nor can install malware. Therefore, even vulnerable environments such as unpatched Windows XP can be protected without being exploited by both zero-day and known exploits. This is especially beneficial to critical infrastructure and SCADA sites whose systems are usually not patched mainly due to the 24/7 operation requirement.

4. AFFAF is a proactive method (i.e. before any actual exploitation happens) that modifies the behaviour of web browser plugins in order to block triggering of exploits embedded in the exploit kits. Therefore, unlike reactive detection techniques, it works well no matter how much exploit kits' JavaScript code is obfuscated.
5. AFFAF is more fine grained than blocking web browser plugins or blacklisting them. This is especially useful to systems where disabling a specific plugin is impossible. For example, some enterprise-wide software solutions require Java [27].
6. Employing AFFAF does not require significant workload for plugin vendors. First, modern browser plugins already provide automatic version check functionality that is reusable in AFFAF that needs the latest version number. Those plugins automatically check the latest version and pop up an update window to users, even though many users do not update them. Second, reporting genuine version number only to the vendor is also simple to implement; browser plugins can use digital certificates or other authentication schemes to verify the subject who is checking their version.
7. The proposed AFFAF technique can be used in conjunction with other techniques that are already in place.

## 4 Implementation

We have implemented a prototype as a separate system between the two implementation methods discussed in the previous section (Fig. 2). Unlike a browser plugin vendor who can apply AFFAF to its product by updating its source code, we had to patch individual plugin's binary if we were to implement AFFAF in the integrated way. Furthermore, having a separate system provides a more efficient way to support more than one web browser plugin as an independent (non-vendor) developer.

*Overview and Scope:* The current implementation uses two techniques: Internet Explorer extension that enables JavaScript injection and object override feature of JavaScript. It demonstrates most merits of AFFAF including defence against browser exploit kits, zero-configuration, and reporting real version numbers only to corresponding vendors. Lastly, it does not affect update procedure of web browser plugins.

The prototype is capable of modifying version numbers of Flash, Adobe Reader, and Internet Explorer. Support for other browsers and plugins like Java can be added in the future. It consists of two parts, a browser extension and JavaScript code, as given in Fig. 3. The former enables the prototype to inject (i.e. pre-load) JavaScript code in every web page before it is rendered, and the latter intercepts and modifies version numbers.

*Pre-loading JavaScript in Web Pages:* We have tested two platforms for JavaScript injection purpose: IE7Pro and Crossrider. IE7Pro is a browser extension for Internet Explorer 6, 7, and 8 that aims to enhance the feature set

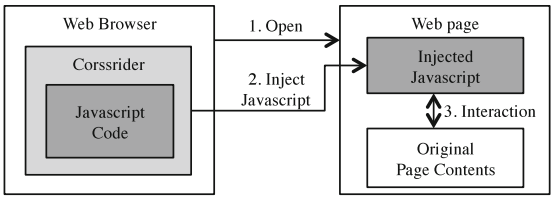


**Fig. 3.** Prototype implementation of `AFFAF`

provided by the browser. Even though IE 9 is not officially supported, IE7Pro is compatible with it as well. IE7Pro adds features such as tab enhancement, advertisement and flash blocker, mouse gestures, inline search, privacy enhancements, online bookmark service, and user script support. We tested the user script functionality to pre-load JavaScript code, and verified its operation. Second option is Crossrider. It is a cloud-based development framework that lets developers quickly and easily create cross browser extensions. When the developers write code using Crossrider APIs and JavaScript, Crossrider builds a multi-browser extension for the code. It supports Internet Explorer, Chrome, Firefox, and Safari. Among various APIs it provides, `includeJS()` and `addInlineJS` add the contents of the specified JavaScript resource to web pages. We used this API to pre-load our `AFFAF` JavaScript, and confirmed it satisfied our requirements. The features such as user script of IE7Pro and API `includeJS()` of Crossrider are mainly intended for page view optimisation such as social media site decluttering. Even though both IE7Pro and Crossrider are viable options, we have selected Crossrider as our JavaScript injector for a couple of reasons. First, Crossrider is actively being developed and supported, while IE7Pro is abandoned and has not been updated since June 2010. Second, building multi-browser extension has more potential than making an IE-only one, even though our current prototype mainly targets IE just like major exploit kits do.

Our Crossrider extension is a shell for `AFFAF`, hence simple as below. Once a user installs this extension, a file (`affaf.js`) is pre-loaded each time a page is viewed, and the included JavaScript code performs `AFFAF`'s functionalities.

*Modifying Version Numbers:* The main body of our implementation is the JavaScript code injected into every web page the browser loads. First, it uses a JavaScript's inheritance feature to override original `ActiveXObject` that is used by developers and attackers when checking plugin version:

```
var f = ActiveXObject;

// Override ActiveXObject
var ActiveXObject = function(progid) {
  var ax = new f(progid);
  this.prototype = ax.prototype;
  ...
```

The overridden object has two functions, one for Flash and the other for Adobe Reader. In the case of Flash, `GetVariable()` is overridden in order to

return fake or real version number depending on the subject that requests the version. The overridden function first gets the actual version of the active Flash plugin, adds some random numbers to its major and minor version numbers, and returns the newly created version number to its caller. It returns the original version number only to "adobe.com". Essentially identical operations are performed for Adobe Reader as well. Only version parsing routine, `progid` string that is compared and the overridden function are different. This implementation also does not affect updates of Flash and Adobe Reader, since each of these two plugins use a separate process to check and update themselves, which is independent of the Internet Explorer. If we modified the actual version number embedded inside plugins, update procedure would have also been affected.

Lastly, in order to modify the version number of Internet Explorer, `navigator` object is redefined using the following JavaScript code[7], because IE (unlike other browsers like Chrome and Firefox) does not allow to override getter functions of `navigator`'s properties. In the redefined `navigator` object, properties related to browser version like `userAgent` and `appName` are replaced with fake strings. And all other properties, such as `systemLanguage` and `cookieEnabled`, are defined as corresponding original values so as to make the new object a complete replacement of the original `navigator`.

```
// navigator redefined
var navigator=new Object;

// userAgent and other properties redefined
navigator.userAgent='Mozilla/5.0 (compatible; ...)';
navigator.platform='Win32';
navigator.appCodeName='Mozilla';
navigator.appName='Microsoft Internet Explorer';
navigator.appVersion='5.0 (compatible; MSIE ...)';
```

Even though this prototype is not a full implementation of `AFFAF`, it achieves its major aspects. First, as a browser extension, it intercepts communications between the web browser and web pages, and injects JavaScript code that modifies version number of Flash and Adobe Reader. Second, it does not affect the update procedures of Flash and Adobe Reader. Third, it checks the hostname of version requesting subject, and returns the real version number only to the vendor (Adobe in this case). We have verified this code using Adobe's Flash version check page[8]. It is also a zero-configuration extension. A user simply needs to install this on their system. In addition, it is a lightweight implementation without any kernel module and dedicated user process.

*Limitations:* Some exploits (in exploit kits) can be promiscuously triggered if they are harmless from the perspective of attackers. For example, an exploit for Internet Explorer (CVE-2006-003) is unconditionally triggered in Blackhole 2.0. Therefore, it is very important to combine `AFFAF` with a reactive detection and prevention solution in order to provide the maximum protection.

[7] This is a simplified representation. For instance, many variables have been omitted whereas some others have been replaced with static strings.

[8] http://helpx.adobe.com/flash-player/kb/find-version-flash-player.html.

There is more than one way of checking versions of Flash and Adobe Reader. For instance, action script inside a Flash object can check the version number of a currently active Flash object. Since our implementation uses JavaScript, it cannot intercept this kind of version checking. It can only hook version checking methods that make use of JavaScript. However, this is an implementation issue and only applies to our prototype, not to the concept of `AFFAF`. When plugin vendors like Adobe and Oracle employ `AFFAF`, it would work in all version checking situations. In addition, as our evaluation results (Sect. 5) suggest, this prototype is enough to invalidate many of currently available exploit kits. We suppose this is because using JavaScript for version checking is the *de-facto* standard among developers and attackers.

Finally, the current prototype implementation does not work for other web browsers like Chrome and plugins such as Java and Microsoft Office. Again, this limitation is applied only to the current version of the prototype, not to the proposed methodology.

## 5 Evaluation

Evaluation of the prototype implementation has been performed from three different aspects. First, `AFFAF` was tested against live or locally deployed exploit kits in order to verify its effectiveness. This evaluation demonstrates the efficacy of the protection that `AFFAF` provides against real exploit kits. Next, we visited a wide range of benign web sites that contained Flash and/or PDF contents, and ascertained whether those sites worked well without any issue. This evaluation is a test for false positive; for the prototype, a failure of Flash or PDF content with its deployment is a false positive. Lastly, we tested the implementation with dedicated browser plugin scanning services to test its robustness.

*Configured Vulnerable Environment:* Our evaluation environment is composed of two VMware virtual machines running on a dedicated PC with a 3.4 GHz Intel Core i7 and 16 GB RAM. One VM runs Windows XP Professional SP3 and the other runs 32-bit Windows 7 Ultimate. Both are equipped with various vulnerable versions of Flash (10.0.45.2 and 11.0.1.152) and Adobe Reader (8.1.0, 9.0.0 and 10.0.1). Versions for each software were decided based on the CVE information available from Table 1 so that they are vulnerable to exploit kits. In total, twelve ($2 \times 2 \times 3$) software configurations were set up. For each configuration, two snapshots are saved: one with the prototype and the other without, resulting in 24 separate snapshots. System utilities like `Procmon` (Process Monitor) and `tcpdump` are also deployed in each VM so as to scrutinise any triggered exploitation. Lastly, no web browser plugins other than Flash and Adobe Reader are installed on the VMs so they are not compromised by other exploits such as targeting Java and Microsoft Office.

### 5.1 Defence Against Exploit Kits

The best way to test `AFFAF`'s effectiveness is to visit real-world exploit kit URLs with vulnerable browser plugin configurations. We have collected exploit kit

**Table 2.** Examples of tested live exploit kits

| Exploit kit | URL | Blocked by AFFAF |
|---|---|---|
| Blackhole 2.0 | ilianorkin.ru:8080/forum/links/column.php | YES |
| Blackhole 2.0 | actsforcharged.com/closest/209tuj2dsljdglsgjwrigslgkjskga.php | YES |
| Blackhole 2.0 | juhajuhaa.ru:8080/forum/links/column.php | YES |
| Blackhole 2.0 | ighjaooru.ru:8080/forum/links/public_version.php | YES |
| Blackhole 2.0 | http://eveningwiththeeditors.com/wp-content/plugins/wp-plugin-repo-stats/wps.php?c002 | YES |
| Blackhole 2.0 | www.quickcraft.com.br/infourl.htm | YES |
| Blackhole 2.0 | hillaryklinton.ru:8080/forum/links/column.php | YES |
| Redkit | senreibehn.narod.ru/ | YES |
| Redkit | actionpreventive.com/mhas.htm?j=1335200 | YES |
| Sakura | oto-drukarnia.pl/wp-content/themes/twentyten/amaz.html | YES |
| Cool | www.appvenue.dk/seoadvertbb.html | YES |
| Bleeding Life 2 | localhost (set up with leaked version) | YES |

URLs from multiple security mailing lists and sites including Malware Domain List[9], URL Query[10], and ZScaler URL Risk Analyzer[11]. In addition to these live URLs, we downloaded a popular exploit kit (Bleeding Life 2), and configured it in our own testbed.

*Evaluation Method:* The evaluation has been conducted in four steps. First, we collected malicious URLs from multiple sources and input them to our 24 VM snapshots. Second, each URL was visited by each snapshot. This process was automated using VMware's script support. Next, we analysed twelve snapshots that do not have `AFFAF` installed in order to check whether all the components of an exploit kit, such as JavaScript libraries and exploit code, are live and active. This was important since many collected landing pages were linked to non-existent exploit code even one day after the pages were disclosed. Through this step, we verified (1) that twelve plugin configurations were actually exploitable by exploit kits and (2) that the exploits for Flash and Adobe Reader included in the exploit kits were active and working. Landing URLs with broken exploit links were excluded in this step for accurate evaluation. As a result, we tested 50 live URLs of working exploit kits. Table 2 shows 11 of them, which are still online and active at the time of August 2013. In the last step, we compared two snapshots of each software configuration in order to check if `AFFAF` prevented exploit kits from triggering their exploits.

*On the Modification of Version Number:* For each snapshot, we made `AFFAF` advertise various versions for Flash and Adobe Reader, and examined if it successfully blocked exploit kits. When modifying version numbers, `AFFAF` only adds (never subtracts) a random number to the actual version number. And the random number was selected in a way that the resultant fake version is same or

[9] http://www.malwaredomainlist.com.
[10] http://urlquery.net.
[11] http://zulu.zscaler.com/.

higher than the latest version. Therefore, at least two is added in the case of Adobe Reader 8 to make it look like 10, and one in the case of Flash 10 to make it seem to be 11. For the versions whose major version number is same as the latest one (e.g. Flash 11), only minor version numbers are modified.

*Results and Discussion:* As shown in Table 2, `AFFAF` successfully blocked all the exploit kits (50 live URLs and one locally deployed exploit kits) evaluated with twelve different software configurations. In other words, 612 cases (12 configurations $\times$ 51 links) that would have been exploited were protected by `AFFAF`. Validation was conducted in two ways. First, we recorded exploit URLs, such as malicious Flash and PDF files, when visiting malicious URLs without `AFFAF`. Then, we analysed packet dumps and verified no such file was downloaded during `AFFAF` test. Because exploit kits download and execute a particular exploit after it decides to use the exploit [3,5,20–22,24,29], the fact that exploit kits did not download actual exploits proves no exploit was triggered. Second, we also examined `Procmon` log files in order to double-check that there was no evidence of exploitation.

All the five exploit kits tested in this evaluation (Blackhole 2.0, Redkit, Sakura, Cool and Bleeding Life 2) are actively updated and maintained at the time of mid 2013 [4]; three of them (Blackhole, Sakura and Bleeding Life) are included in the most popular exploit kits [1]. This implies that (1) `AFFAF` is effective in blocking major exploit kits, and (2) most exploit kits use JavaScript for version detection, as the current prototype implementation is only capable of intercepting JavaScript-based version checking methods. Indeed, we found that all of them use a same public library called `PluginDetect`[12] for browser plugin detection. `PluginDetect` is a JavaScript library that detects browser plugins. It is intended to work with all the major browsers such as Internet Explorer, Firefox, Mozilla, Netscape, Chrome, Safari, Opera, SeaMonkey and Flock. We further investigated this interesting aspect, and it turned out that `PluginDetect` is also used in other exploit kits such as Neutrino 2.0, Nuclear [30] and Whitehole [29]. It seems that there are a couple of plain advantages for exploit kit developers to use publicly available library: higher anonymity and better version detection. For example, custom version detection code can be used as a signature/feature for a particular exploit kit. More importantly, it may contain bugs inside its custom version checking routine. Lastly, defensive solutions like anti-virus and IPS cannot detect `PluginDetect` since benign web sites also use it for compatibility check purpose.

It should be noted that many of the exploits in Table 1 were zero-days when they were first employed in exploit kits. This was achieved in our testbed VMs using outdated plugins. This demonstrated that `AFFAF` is effective in blocking zero-day exploits as well as known ones.

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

### 5.2 Benign Web Sites

We also evaluated AFFAF on benign web sites in order to measure its false positive rate. For AFFAF, a false positive means that a legitimate web site does not work properly under the deployment of AFFAF.

*Alexa Top 100 Web Sites and Embedded PDF Contents:* First, Alexa top 100 web sites including Flash-centric ones such as YouTube, Dailymotion, youku, LiveJasmin, ESPN, CNN, and CNET.com were tested. In order to verify that Flash contents actually work, not only the first pages but also specific pages containing Flash-based contents were visited. We confirmed all the Flash contents on those web sites worked correctly, which implies AFFAF's false positive rate is virtually zero. For PDF contents, we searched PDF files and PDF-embedded web pages on Google, visited them, and checked if PDFs are displayed correctly. Adobe Reader plugin worked well in all the test cases. This is obvious since actual content-embedding code (e.g. <object> tag) is not affected by the prototype implementation. Only version checking APIs like GetVariable() and GetVersions() are intercepted and overridden.

*Top 10 Flash-based Web Sites:* Most of the Flash contents tested with Alexa top 100 web sites are video and advertisement. Even though Flash is basically a video container, it can also be used for developing an entire web application such as online game and graphic editor. In order to compensate for this potential limitation, we performed a second evaluation with the top ten Flash web sites selected by eBiz[13]. These ten sites include a variety of Flash-based web sites from a driving game and to a museum virtual tour. Again, we verified that all the Flash-based sites worked without any error. This experiment reaffirms AFFAF's false positive rate is zero.

### 5.3 Browser Scan Services

As exploit kits are one of the major cyber threats, security companies like Rapid7 (well-known for Metasploit) and Qualys provide web service, which checks versions of web browser plugins and warns users of outdated ones. We have tested AFFAF with two browser scanning services, one from Rapid7[14] and the other from Qualys[15]. The purpose of this evaluation is to check AFFAF's robustness. We verified that both services reported fake version numbers returned by AFFAF for Flash and Adobe Reader, while correct versions were detected for other plugins like Silverlight. This means AFFAF works well even against dedicated plugin scanning services.

[13] http://www.ebizmba.com/articles/best-flash-sites.
[14] http://browserscan.rapid7.com/scanme.
[15] https://browsercheck.qualys.com.

## 6 Related Work

There have been many research efforts to detect and prevent drive-by download attacks and exploit kits. Some are focused on malicious host detection, whereas others are on language-specific detection.

*Malicious Host/Download Detection:* Lu *et al.* [6] proposed a browser independent operating system kernel extension designed to eliminate drive-by malware installation performed by exploit kits. Li *et al.* [15] developed a topology-based malicious host detection technique based on the unique feature they found by studying a set of topologically dedicated hosts discovered from malicious web infrastructures. Invernizzi *et al.* [10] suggested an approach to search the web more efficiently for pages that are likely to be malicious. Their system leverages the crawling infrastructure of search engines to retrieve URLs that are much more likely to be malicious than a random page on the web by starting from an initial seed of known malicious web pages. Canali *et al.* [8] suggested a filter that quickly discards benign web pages and detects malicious content using static analysis techniques. Wang *et al.* [17] developed an automated web patrol system to automatically identify and monitor these malicious sites. Nappa *et al.* [18] studied drive-by download operations and proposed a technique to detect exploit servers managed by the same organisation.

*Language-specific Detection:* On detection of malicious JavaScript code, Kapravelos *et al.* [7] presented an automatic detection technique for evasive JavaScript code. It used the observation that two scripts that are similar should be classified in the same way by web malware detectors. Curtsinger *et al.* [11] used Bayesian classification of hierarchical features of the JavaScript abstract syntax tree to identify syntax elements that are highly predictive of malware. Cova *et al.* [12] combined anomaly detection with emulation to identify automatically malicious JavaScript code and to support its analysis. Kolbitsch *et al.* [9] proposed a JavaScript multi-execution virtual machine as a way to explore multiple execution paths within a single execution so that environment-specific malware reveals itself. Rieck *et al.* [31] embedded an automatic drive-by download detection and prevention system inside a web proxy, and blocked delivery of malicious JavaScript code using static and dynamic code features. Nikiforakis *et al.* [32] performed a large-scale crawl on the Internet and suggested a set of metrics that can be used for JavaScript provider assessment. Through this process, they detected four new types of vulnerabilities. Regarding Java-based malware, Schlumberger *et al.* [33] proposed a detection system for malicious Java applet based on static code analysis.

## 7 Concluding Remarks

In this paper, we introduced `AFFAF`, which is a new approach to protecting vulnerable systems from a prevalent cyber threat, namely the exploit kits. It is a

proactive methodology that blocks the execution of exploit kits using the version information of the browser plugins. It is a zero-configuration solution from user's perspective, and is an easy-to-employ method from developer's view. We have implemented a lightweight prototype, and demonstrated that `AFFAF` provided protection to vulnerable environments with outdated plugins by validating it against 50 real-world and one locally deployed exploit kit URLs. Tested exploit kits included popular and well-maintained ones such as Blackhole 2.0, Redkit, Sakura, Cool and Bleeding Life 2. Also, we showed that the false positive rate of `AFFAF` is virtually zero, and the technique is robust enough to be effective against real web browser plugin scanners.

Currently we are continuing to test `AFFAF` against new coming and live exploit kits that will be armed with new zero-day exploits, and confirming that `AFFAF` is still effective on those future threats. Support for other browsers and plugins like Java can be added in the future version of our `AFFAF` prototype. In addition, the concept of `AFFAF` can be extended to any type of software, thus applying it to other categories of software such as operating system can be part of our future work.

## References

1. Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C.J., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsillidis, A.: Manufacturing compromise: the emergence of exploit-as-a-service. In: CCS 2012, Raleigh, North Carolina, USA (2012)
2. Fossi, M., Egan, G., Johnson, E., Mack, T., Adams, T., Blackbird, J., Graveland, B., McKinney, D.: Symantec report on attack kits and malicious websites. Technical report (2011)
3. Cannell, J.: Tools of the Trade: Exploit Kits, February 2013. http://blog.malwarebytes.org/intelligence/2013/02/tools-of-the-trade-exploit-kits/
4. contagio: An Overview of Exploit Packs (Update 19.1), April 2013. http://contagiodump.blogspot.com
5. Jones, J.: The State of Web Exploit Kits. Black Hat USA, Las Vegas, Nevada, USA (2012)
6. Lu, L., Yegneswaran, V., Porras, P., Lee, W.: Blade: an attack-agnostic approach for preventing drive-by malware infections. In: CCS 2010, Chicago, Illinois, USA (2010)
7. Kapravelos, A., Shoshitaishvili, Y., Cova, M., Kruegel, C., Vigna, G.: Revolver: an automated approach to the detection of evasive web-based malware. In: 22nd USENIX Security Symposium, Washington, D.C., USA, August 2013
8. Canali, D., Cova, M., Vigna, G., Kruegel, C.: Prophiler: a fast filter for the large-scale detection of malicious web pages. In: WWW 2011, Hyderabad, India (2011)
9. Kolbitsch, C., Livshits, B., Zorn, B., Seifert, C.: Rozzle: De-cloaking Internet malware. In: IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA (2012)
10. Invernizzi, L., Comparetti, P.M., Benvenuti, S., Kruegel, C., Cova, M., Vigna, G.: EVILSEED: a guided approach to finding malicious web pages. In: IEEE Security and Privacy, San Francisco, CA, USA (2012)

11. Curtsinger, C., Livshits, B., Zorn, B.G., Seifert, C.: ZOZZLE: fast and precise in-browser javascript malware detection. In: USENIX Security 2011, San Francisco, CA, USA (2011)
12. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attacks and malicious JavaScript code. In: WWW 2010, Raleigh, North Carolina, USA (2010)
13. Richards, J.: Dangerous Drive-by Downloads: Protecting yourself with NoScript, September 2012. http://cmu95752.wordpress.com/2012/09/27/dangerous-drive-by-downloads-protecting-yourself-with-noscript/
14. Ducklin, P.: Apple bans outdated Adobe Flash plugins from Safari, March 2013. http://nakedsecurity.sophos.com/2013/03/04/apple-bans-oudated-adobe-flash-plugins-from-safari/
15. Li, Z., Alrwais, S., Xie, Y., Yu, F., Wang, X.: Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In: IEEE Symposium on Security and Privacy (S&P) 2013, Berkeley, CA, USA (2013)
16. Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., Feamster, N.: Building a dynamic reputation system for DNS. In: USENIX Security 2010: Proceedings of the 19th USENIX Conference on Security, August 2010
17. Wang, Y.M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.: Automated web patrol with strider honeymonkeys. In: Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA (2006)
18. Nappa, A., Rafique, M.Z., Caballero, J.: Driving in the cloud: an analysis of drive-by download operations and abuse reporting. In: Rieck, K., Stewin, P., Seifert, J.-P. (eds.) DIMVA 2013. LNCS, vol. 7967, pp. 1–20. Springer, Heidelberg (2013)
19. Rajab, M., Ballard, L., Jagpal, N., Mavrommatis, P., Nojiri, D., Provos, N., Schmidt, L.: Trends in circumventing web-malware detection. Technical report (2011)
20. Oliver, J., Cheng, S., Manly, L., Zhu, J., Dela Paz, R., Sioting, S., Leopando, J.: Blackhole exploit kit: a spam campaign. Not a Series of Individual Spam Runs, Technical report (2012)
21. Desai, D., Haq, T.: Blackhole exploit kit: rise & evolution. Technical report, September 2012
22. Mieres, J.: Phoenix exploit's kit from the mythology to a criminal business. Technical report, August 2010
23. Kotov, V., Massacci, F.: Anatomy of exploit kits: preliminary analysis of exploit kits as software artefacts. In: Jürjens, J., Livshits, B., Scandariato, R. (eds.) ESSoS 2013. LNCS, vol. 7781, pp. 181–196. Springer, Heidelberg (2013)
24. Sood, A.K., Enbody, R.J.: Browser exploit packs - exploitation tactics. In: Virus Bulletin Conference, Barcelona, Spain, October 2011
25. Higgins, K.J.: No Java Patch For You: 93 Percent of Users Run Older Versions of the App, June 2013. http://www.darkreading.com/vulnerability/no-java-patch-for-you-93-percent-of-user/240156053
26. Rashid, F.Y.: Most Adobe Reader Users Running Outdated, Unpatched Versions, July 2011. http://www.eweek.com/c/a/Messaging-and-Collaboration/Most-Adobe-Reader-Users-Running-Outdated-Unpatched-Versions-213010/
27. Bit9: java vulnerabilities: write once, pwn anywhere. Technical report (2013)
28. Mozilla support: Outdated Adobe Acrobat plugin, March 2013. http://support.mozilla.org/en-US/questions/953805
29. Chua, J.P.: Whitehole Exploit Kit Emerges, February 2013. http://blog.trendmicro.com/trendlabs-security-intelligence/whitehole-exploit-kit-emerges/

30. wmetcalf: Monthly Archives, May 2013. http://www.emergingthreats.net/2013/05/
31. Rieck, K., Krueger, T., Dewald, A.: Cujo: efficient detection and prevention of drive-by-download attacks. In: ACSAC 2010, Austin, Texas, USA (2010)
32. Nikiforakis, N., Invernizzi, L., Kapravelos, A., Van Acker, S., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: You are what you include: large-scale evaluation of remote javascript inclusions. In: CCS 2012, Raleigh, North Carolina, USA (2012)
33. Schlumberger, J., Kruegel, C., Vigna, G.: Jarhead analysis and detection of malicious Java applets. In: ACSAC 2012, Orlando, Florida, USA (2012)