Weihang Wang

My research goal is to make web applications secure and sustainable through program analysis and transformation. As a primary source of income for many Internet companies, web advertising (ads) has a profound impact on the security and sustainability of today’s web ecosystem. Web advertising, however, is extremely complex, highly vulnerable to attacks, and prone to bugs. My research has been focused on creating effective tools to improve the security and sustainability of real-world web applications [1, 2, 3, 4].

1. Research Highlights

Web advertising is a multi-billion-dollar industry. Under the hood, a gigantic web advertising system connects various parties: websites join the system as publishers and offer ad slots to ad networks/exchanges, which further conduct bidding auctions to sell the ads to advertisers or resell them to another network/exchange. Before the final ad content is delivered to a website visitor, an ad slot may go through an extremely intricate chain of bidding and reselling transactions that involve more than 50 ad networks. These various networks are physically scattered across the globe which makes them particularly enticing to cybercriminals. Similar to normal advertisers, cybercriminals sign up with an ad network and then bid in real time. However, instead of displaying ads, they deliver a malicious payload to website visitors which exploits vulnerabilities in browsers or browser plug-ins. Once malicious ads infiltrate a visitor’s browser, the malware hidden within these ads can execute a number of harmful attacks including dropping ransomware, spyware, or anything that benefits the attackers with little to no user interaction required. To reach a wide range of visitors, malicious advertising (malvertising) attacks purposefully target legitimate websites with high traffic. A massive malvertising attack named AdGholas spotted in 2016 compromised high-profile sites, including msn.com, exposing 43 million visitors to malware within just 24 hours [5]. According to a report issued by Ernst & Young in 2015, the damage caused by malvertising is striking: each year, the U.S. web advertising industry loses $1.1 billion to malvertising [6]. In addition, dynamically including scripts from ad networks all over the world often leads to severe deteriorations in user experience because third-party ad scripts may include buggy code that consumes excessive computational resources. Such buggy scripts drop down page rendering speed and/or introduce unnecessary delays in user-perceived response time. For example, we found that defective ad scripts on popular websites such as chicagotribune.com and accuweather.com use up the CPU quickly and make the entire browser unresponsive within 1 minute [4].

Needless to say, concerns over malvertising and user experience deteriorations are driving visitors to seek shelter behind the wall of ad-blocking software which blocks ads based on pre-defined signatures. As a temporary remedy for the scourge of malvertising, ad blockers, unfortunately, are only able to prevent malicious ads where damage has already occurred. Additionally, despite the fact that browsers generally render pages faster with all ads blocked, in the long run blocking ads will devastate the entire web ecosystem by destroying the business model many Internet companies rely on and depriving publishers of vital ad revenue. Without the monetary revenue gained from ads, various web companies would go out of business, which is to no one’s benefit. A survey by PageFair, an ad-blocking measurement company, suggests that the estimated loss of global revenue due to blocked advertising during 2015 was $21.8 billion [7].

My research focuses on developing novel techniques to solve the problems that advertisements are malicious, abusing resources or undesirably blocked. More specifically, I have proposed a novel programming support system for publisher developers to program their regulation logic on ads. I have developed an effective technique, called PAD, for malicious ads prevention via JavaScript memory randomization [1]. PAD effectively suppresses a large set of malvertising cases which have infected high profile websites such as yahoo.com, msn.com, answers.com, etc., all of which have millions of daily visitors. PAD provides a general protection that makes it not only effective in stopping malicious ads with known vulnerabilities, but also preventing malicious ads that exploit zero-day vulnerabilities. To mitigate deteriorated user experience caused by buggy ads, I have proposed AdJust which allows publishers to regulate resource consumption by ads via runtime events scheduling [4]. AdJust effectively mitigates performance degradations incurred by ads and improves the load time of page content by 2.2 times. In an effort to thwart the ever-increasing use of ad-blocking software that destroys the economic structure of the advertising ecosystem, I have built a practical system, WebRanz, for circumventing ad-blocking software through web page randomization to make web applications sustainable [2]. WebRanz is able to circumvent the state-of-the-art ad blockers on a large number of popular websites while retaining the faithful appearances

http://www.cs.purdue.edu/homes/wang1315/
and functionalities of these websites. Interestingly, within months of publishing WebRanz, Facebook and other websites used the same approach to protect against ad blockers and thus retain their ad revenue [8].

In addition, modern browsers employ a highly concurrent execution model which leads to web concurrency bugs. These bugs can cause severe problems including permanent data corruption on servers, denial of service, privilege escalation, etc. My research has introduced an innovative scheme that leverages constraint-solving to automatically fix concurrency bugs in a safe and cost-effective fashion [3]. ARROW has fixed 151 races from 20 real-world commercial websites.

2. Research Projects

Next, I will discuss more details of my research projects.

2.1 Web Advertising

The complexity of the ad ecosystem presents several challenging issues: (1) Third-party scripts may include malicious and buggy code that publishers have no control over; (2) dynamically including scripts from all over the world makes pattern-based security checkers and regulators not amenable to identifying ads violations; (3) multiple layers of delegations create isolations and set apart any protection in one layer from another. My research is the first to address these challenges by providing a persistent runtime programming support for publisher developers to effectively protect against malicious ads, buggy ads that degrade user experience, and undesirable ad blocking [1, 4, 2].

Advertisements Are Malicious

As a top security threat that continues to develop, malvertising keeps slipping through the cracks because none of the stakeholders involved in advertising (i.e., advertisers, publishers, and ad networks) is picking up the slack. Among the various parties involved, advertisers have no interest in protecting publishers’ legitimate rights. Though ad networks strive to suppress malicious ads with pattern-based gatekeeping, they are limited to attacks with known tactics. I consider publishers to be the ideal party to protect themselves from malicious ads. To do so, I have proposed a novel programming support system, called PAD, that allows publishers to program their protection logic of secure advertising [1]. Due to the unique virtualized execution model of web browsers, cybercriminals have to inject malicious payloads via JavaScript interfaces that can manipulate consecutive memory regions. To corrupt malicious payloads and thus neutralize attacks, PAD randomizes values in memory. In particular, PAD encodes values written to memory and decodes them upon retrieval using the same key employed in encoding. In this way, malicious payloads are encoded before being injected into memory. Since malicious payloads are always executed natively, without going through the JavaScript interfaces, malicious payloads are not properly decoded before execution and hence broken. In contrast, benign data used inside JavaScript is decoded properly as it has to go through the JavaScript interfaces. PAD also provides a novel persistent runtime, which can self-install in each delegation layer and ensures persistent protection throughout the entire life-cycle of ad delivery. With a negligible runtime overhead (1.67%) on 200 Alexa websites, PAD effectively prevented a large set of real incidents of malvertising attacks.

Advertisements Are Abusing Resources

To understand how ads use computational resources and their effects on user experience, I studied the Alexa top 200 news websites that contain ads and observed two common issues caused by ads: performance degradation such as substantial delays in waiting for publisher content or delays in interacting with the site, and priority inversion including the inverted loading order between ads and publisher content or that among two different ads. Priority inversion occurs when lower priority content is loaded before higher priority content which inverts the priority order of the two. For example, we consider off-screen ads to have lower priority than ads in the current view. Loading the lower priority off-screen ads may delay ads in the current view, which creates a priority inversion that conflicts with the developer’s intention. To address the issues mentioned above, I have proposed an effective technique, AdJust, that allows publishers to specify and adjust the frequency constraints on events associated with third-party ads [4]. By setting frequency caps on various events, such as HTTP requests,
script load, DOM creations, timers and exceptions, AdJust mitigates user experience degradations and enforces consistent ads experience to all users. The results on the Alexa top 200 news sites are encouraging: AdJust effectively mitigated degradations that freeze web browsers (on 36 websites), reduced the wait time for publisher content (on 61 websites), prioritized publisher content (on 166 websites) and ensured consistent orders among top ads (on 68 websites).

Advertisements Are Undesirably Blocked

With the rise of ad-blocking software, ad-supported Internet services would eventually vanish. To maintain a sustainable web ecosystem, I developed a novel, lightweight web page randomization technique, called WebRanz, that circumvents ad-blocking using a content (including HTML, CSS, and JavaScript) randomization technique [2]. Particularly, WebRanz randomizes page elements’ signatures such as URL, id, and classname to invalidate the predefined patterns that ad blockers use to filter out ads, and de-randomizes elements when they are accessed through interface functions. To ensure page appearances and legitimate functionalities are intact, WebRanz was designed to preserve the dependencies between DOM objects and CSS selectors, between DOM elements and JavaScript, and to be able to handle dynamically generated elements as well as resolve randomized URLs. WebRanz effectively prevents ad blockers from blocking ads on 221 Alexa top web pages. It has a negligible runtime overhead (1.38%) which does not downgrade user experience. Finally, WebRanz also benefits the defense against web bots that rely on predefined patterns to locate DOM elements. Our experiments showed that WebRanz successfully prevented representative web bots targeted at popular websites including Amazon, Groupon, Twitter, and Yelp.

2.2 Web Concurrency Bugs

Prior to my work on protecting the web advertising ecosystem, I worked on the automatic repair of web concurrency bugs which can cause permanent data corruptions, privilege escalations, and usability issues. For example, web application races on a cloud service page cause users’ files to be undesirably deleted from the server [9]. I have developed ARROW, the first technique that can automatically fix race conditions on client side pages based on static analysis [3]. In particular, ARROW statically models a web page as a causal graph denoting happens-before relations between elements. ARROW then detects races by identifying inconsistencies between the graph and the dependence relations intended by the developer. To ensure consistencies, ARROW leverages a constraint solver to compute additional happens-before edges. The page is then transformed to ensure the new happens-before relation by re-ordering the elements in the page or adding customized synchronizations to respect the repair edges. ARROW effectively fixed 151 races from 20 real-world commercial websites.

2.3 Other Projects

I have developed ZoomHalo, a human-computer interaction technique for comparing the overview and details within different parts of a visual representation simultaneously [10]. I have also been involved in a number of research collaborations in model-based casualty inference for attack investigation [11], attacks on retargeting advertising and neural networks [12, 13], cross-platform binary code reuse [14], and subscription algorithms in publish/subscribe networks [15].

3. Future Research Agenda

The goal of my research is to integrate program analysis and software engineering techniques with the development of secure and sustainable web applications. In pursuing this goal, my future research plan is as follows:

Ensuring Brand Safety for Advertisers and Publishers

In recent years, advertisers have developed a pronounced concern for the safety of their brands. With the rise of fake news and inappropriate extremist views making headlines, advertisers more than ever are concerned about their ads appear on any website containing content that can potentially damage advertisers’ brand reputation. In March 2017, major brands including Verizon and Walmart pulled their ads after they were found to be
appearing alongside YouTube videos promoting terrorism and hate speech [16]. YouTube allows advertisers to pick the types of videos they would like to be associated with based on keywords. Yet it is very challenging to keep disagreeable content completely quarantined from advertisers. Brand safety, however, concerns not only advertisers, but also publishers. Between 2015 and 2016, various Russian entities attempted to influence the 2016 United States presidential election by placing political ads on platforms such as Facebook, Twitter, and Google. To fight election interference, in 2017, Facebook removed 470 fake accounts believed to be linked to the Russian government that ran political ads [17]. Relying on advertisers and publishers to proactively disclose whether they provide appropriate content cannot stop the miscreants. To this end, I plan to build programming tools that leverage machine intelligence to understand the ads and contextualize website content, ensuring brand safety for both advertisers and publishers.

Enforcing Security Policies on the Client Side Automatically

Modern browsers enforce numerous security policies to mitigate certain types of attacks by limiting a website’s behavior to what is expected. For security policies to take effect, besides the enforcement by browsers, web applications often need to be implemented to follow a specific design pattern. For example, to prevent executing code from untrusted source, Content Security Policy (CSP) blocks calls to functions which evaluate JavaScript code represented as a string, such as `eval()` and the `Function` constructor. To ensure functionalities are not broken after deploying CSP, the web application developer needs to rewrite the program and avoid using string arguments. Such program transformation is tedious and error-prone, especially when the website is complex and includes many pages. Sometimes, many web application developers may not even be aware of such requirements. Studies on the large number of policies are needed to understand how they are being used and why they are not effective. Additionally, to bridge the gap between browser enforcement and website deployment, I plan to build automated transformation tools for client-side web application policy enforcement by leveraging program analysis techniques. I envision such large scale studies and programming supports to benefit the understanding and correct deployment of security policies on the client side.

Offloading Heavy Transactions of Programmatic Advertising to the Server Side

Despite the low-cost benefits brought by programmatic advertising, the current practice that deploying the entire technology on the client side causes many problems. On the client side, bootstrapping ad libraries are executed to collect visitors’ profiles for bidding auctions. The browser then makes requests to ad networks/advertisers to incorporate bidding and reselling transactions. After the actual content of an ad is loaded, various tracking scripts on the website make additional requests every few seconds to report back to tracking beacons about visitors’ behaviors. During this entire process, the many layers of bidding and reselling generate a large number of requests to ad networks/advertisers, leading to a very poor user experience. For example, a typical American news website makes on average 500 requests finishing in around 17 seconds [18]. Moreover, the ads formats and standards employed by the intermediate servers may not be supported on particular devices. To address these challenges, I plan to decompose the programmatic advertising process, separate the heavy lifting bidding and reselling from other parts of the process and move them to the server side. In WebRanz, we leveraged a transparent proxy for URL randomization/de-randomization. The negligible runtime overhead WebRanz incurred on the proxy server encourages us to explore the feasibility of this solution for boosting user experience and ensuring compatible ads.

References


W. Wang, S. Badam, and N. Elmqvist, “Voronoi-Based Screen Zooming for Structured Visualization Spaces,” under double-blind review.


