**Magecart**

**Origins :**

In early December 2013, we observed attackers targeting and compromising *Magento*PHP scripts en masse for the first time. The modified scripts pulled personal and card data from checkout forms and dumped it to drop servers such as java-e-shop[.]com. This activity continued through 2015 and evolved with some simple obfuscation and data exfiltration via email [5]. Another skimmer, called Visbot, also emerged at this time. It limited its targets to those running Visvo implementations, and bundled stolen data into fake image files stored on the server for the attacker to retrieve later.

The Magecart threat as we know it to date grew out of a single group’s activities that started in 2014 when it began compromising vendor websites and injecting web-skimmers. Several thousand stores were affected during that time. A new group emerged in 2016, with a skimmer and infrastructure distinct from the first group. Both the evolution of skimmers and the multiplication of groups continue to this day. Some of these groups cast wide nets and hit as many vendors as possible. Some carefully conceal their skimmer. Some target third parties to gain access to the thousands of vendors they serve. Some limit their victims to just a few high-value organizations and use specially tailored skimmers, domains and attacks against them. The threat actors continue to grow, evolve and learn.

**Introduction :**

You come across the perfect checkout form and proceed to input your credit card details in order to pay. You are fairly tech savvy and in your mind, you have no doubt that this is a safe, legitimate website, you noticed that the web browser is displaying the usual green padlock icon, which you know indicates that the site is being served securely. You are certain that you are not visiting a fraudulent website. You submit the form, receive your order confirmation, and end your session.

Not long after you receive your most recent credit card statement in the mail and notice some suspicious-looking transactions, purchases you definitely didn’t make. But, how could this be ? and how were your details stolen ?

British Airways website had been compromised with what is known as a *JavaScript skimmer*,  
a malicious script that is added onto real websites to secretly steal customers’ information. In fact, this exact scenario took place in the summer of 2018 when **attackers went undetected for over two weeks** after successfully inserting a JavaScript skimmer on the  
British Airways website, with current estimates of up to 380,000 customers having their  
private information stolen.

The purpose of a JavaScript skimmer is to exfiltrate sensitive user information entered on a legitimate website, such as credit card information, without arousing suspicion from the user; In addition to the intended recipient, your private data is secretly sent to a malicious thirdparty.

These so-called skimming attacks are sometimes referred to *Magecart* attacks owing to the fact that they were first widely identified in use on sites running the Magento software.

**Motivation :**

The number of ‘high-value’ attacks is also on the increase as attackers move away from  
targeting smaller online shops running off-the-shelf e-commerce platforms such as Magento and instead towards larger websites with more potential for data theft due to a greater number of purchases being made but with more difficulty to breach due to using proprietary or hardened systems. In 2018 alone, attackers used JavaScript skimmers to steal user information from the websites of airline British Airways, ticket broker Ticketmaster, push notification service Feedify, hardware retailer Newegg, and optical retailer Vision Direct, to name just a few.

Not only do skimming attacks put consumers at risk of fraud, but there is also a significant  
incentive for businesses to avoid becoming victims of this type of attack too.

As a result of their data breach in 2018, British Airways received a group action lawsuit  
from law firm SPG Law which could mean the airline has to pay out upwards of $500  
million in compensation to affected customers. Penalties for GDPR violations can be up to  
e20 million, or 4% of annual global turnover which has the potential to be financially  
crippling for a business of any size.

**Background :**

**Injection Method :**

In order to carry out a skimming attack, some method of injecting code into a target  
website is required such that the skimmer script can be added onto the web page in the  
first place. There is no single way of doing this and it depends on the how the target  
website is architected and the underlying software it uses. Skimmers may be injected  
directly onto the target website’s server **by a rogue employee** from within the company  
or **by exploiting a vulnerability in an e-commerce platform** such as a cross-site scripting  
**(XSS) attack**. Skimmers could also be inserted onto a site by exploiting a third-party  
resource in what is known as a supply-chain attack; For example, if a JavaScript library  
hosted on a CDN was exploited by way of a skimmer inserted into its code, any website  
which imported the resource would be vulnerable to a skimming attack. In the case of  
the British Airways attack, it is still not publicly known how the skimming script was  
included on the checkout page in the first place, although several online commentators  
speculate that it could be due to a vulnerability in their content management system.  
Further, in October 2018 at least 20 Magento extensions1 were revealed to contain zero-day  
vulnerabilities (i.e. weaknesses that have not previously been discovered or published) that  
gave attackers the ability to inject code into other websites using PHP Object Injection.

**Attack Families :**

Skimming scripts come in many forms and are written and distributed by many different  
hacking groups worldwide, with some clearly inspired by existing skimmers, while others  
are completely unique. For this reason, it is difficult to separate and categorize different  
attack families into a definitive taxonomy. According to California-based cyber security  
firm RiskIQ, there are at least seven distinct forms of JavaScript skimming attacks which  
vary not only by technical implementation but also by the types of site targeted and the  
tactics the hackers use, with a further five variants not currently published publicly. On  
the other hand, Russian cyber-defence firm Group-IB, claims to have found upwards of  
38 unique skimmer families. However, these numbers are growing as more attackers  
begin to utilise skimming attacks, and others create new ways to target their victims and  
avoid detection from security researchers.

**Group 1 & 2 Skimmers :**

Group 1 skimmers were one of the earliest variants found in use, having been first identified  
in 2015. They looking for flaws in common software that the websites use, such as e-commerce software like Magento.

The code works by first examining the URL of the page to check if it is running on a checkout or payment page if so, it will select all input fields on the page (including fields of interest such as credit card number, expiration date, and CVC/CVV [14]), before sending off the data via an AJAX request to the attacker-owned website, sometimes know as a *drop server*. The data is serialized into a query string with each input name acting as the key and the input contents as the value. In the example shown, the data is sent in plaintext without any prior encryption or encoding. It will repeat this process at regular intervals, for example once every five seconds.

The skimmer contains the following elements:  
1. A timeout function is set to 5000. After the timeout period has elapsed, the skimmer will restart its functionality and run through the script again.  
2. The skimmer checks to see if it is on a payments page and inspects specific payment form fields to determine if data has been entered.  
3. If payment data has been entered into the form, it is extracted.  
4. The skimmed data is sent to the drop server at js-save.link via a POST request. In this case, the drop server is the same as that serving the skimmer script to the compromised website.

In this skimmer version, the drop server was the same as the one serving the skimmer. Though it has evolved over the years, tailored by other groups to better fit their needs, the basic elements of the skimmer are still in use.

**Group 3 Skimmer :**

Group 3’s skimmer checks if any of the forms on that page hold payment information.

The list contains a set of field IDs per item which map to a certain payment form. What is interesting about this approach is that while it has some generic input field names such as cardNumber, it also has specific filters for known payment-processing companies. This allows the skimmer to be website agnostic and identify any payment field the skimmer encounters.

The skimmer executes every 700 milliseconds and goes through three steps of data collection.

1. Check if there is a generic form that contains billing in its name. If so, it will extract the billing information from that form and store it in the local storage of the browser under the key \_\_billing123.

2. Check if there is a generic form that contains shipping in its name. If so, it will extract the shipping information from that form and store it in the local storage of the browser under the key \_\_shipping123.

3. Check if any form matches any of the payment form field names in the list discussed above. If there is a match the information is extracted.

The final step is exfiltrating the skimmed data. The data from all three steps is concatenated into one large JSON object. This data is then sent to the drop server in a POST request, with the body of the request containing the stolen data formatted as JSON.

**Group 4 Skimmer :**