**RiskIQ Marks the Macy’s Breach as a Momentous Event in Magecart Attacks**

*The company’s analysis shows a pivot to targeted, customized skimmers*

RiskIQ sheds light on the breach of the Macy's e-commerce platform, which may offer vital clues about the future of Magecart attacks.

Macy's announced the breach on November 14th via [a notice](https://www.documentcloud.org/documents/6552530-MACY-S-NOTICE-OF-DATA-BREACH.html) to customers but did not provide any analysis of the skimmer. According to RiskIQ, which has unique visibility into the JavaScript running on e-commerce sites via its global network of crawlers and sensors, the Macy's incident could represent a pivot to more targeted attacks custom-built for the website functionality of selected targets.

Unlike more basic attacks that cause the majority of Magecart breaches and involve simple, often over-the-counter skimmers deployed to land on a wide range of targets, the attack on Macy's was highly customized. In this attack, the skimmer was purpose-built to exploit the distinct construction of the Macy's website to skim many different types of data. The skimmer intercepted and exfiltrated payment information, the target of most Magecart attacks, but also shipping data saved to customers' personal Macy's accounts. The theft of payment data outside of the checkout process was rarely seen in the wild before this attack.

"The nature of this attack, including the makeup of the skimmer and the skills of the operatives, was truly unique," said RiskIQ Head Researcher Yonathan Klijnsma. "I've never seen a skimmer so meticulously constructed and able to play to the functionality of the target website.

Payment information combined with shipping and billing information creates a valuable package known on the black market as "fullz."

**Other insights in RiskIQ's blog post include:**

* The attack affected customers from October 7th through October 15th and included both the theft of payment information and general PII.
* The compromise was internal—someone had access to files on the webserver and strategically picked a file that would load onto the checkout and Macy's Wallet pages.
* There were two targets for this skimmer, the checkout page and the Macy's Wallet, a web interface in which customers can manage their payment methods. Because of how well-integrated this skimmer was, it could skim the data of users managing their payment methods in their Macy's Wallet accounts.
* The ability for attackers to skim the Macy's Wallet page is a momentous development for web skimming. Before this attack, having stored payment information was an effective way of avoiding skimming attacks.
* These Magecart operatives are highly experienced but don't map to any known Magecart Group. We have yet to see stolen data from this breach go up for sale anywhere as of this writing.

RiskIQ, which first reported on Magecart and broke watershed breaches such as those of [British Airways](https://www.riskiq.com/blog/labs/magecart-british-airways-breach/) and [Ticketmaster](https://www.riskiq.com/blog/labs/magecart-ticketmaster-breach/), continues to track magecart as it evolves, cooperating with victims, law enforcement, and researchers from across the cybersecurity community.

Magecart on 34th Street: Why the Macy's Breach May Define the Next Generation of Attacks

On November 14th, Macy's issued [a notice](https://www.documentcloud.org/documents/6552530-MACY-S-NOTICE-OF-DATA-BREACH.html) to its customers disclosing a breach of its e-commerce platform. Yet another case of the infamous web-skimming syndicate Magecart, the attack affected customers from October 7th through October 15th and included the theft of payment information and general PII. In this article, we will investigate the breach from RiskIQ's perspective and provide new insights into these events based on our unique data, explaining how a sophisticated, meticulously planned breach compromised the Macy's website during the busiest online shopping season of the year.

It's important to emphasize how well-planned and thorough this attack was. Magecart operatives spent a tremendous amount of time learning Macy's website's checkout process and customer journey. Ultimately, their goal was to customize their skimmer to integrate seamlessly into Macy's e-commerce platform to skim information as efficiently as possible while staying undetected for as long as possible.

Finding the Target: It’s All About Placement

The success of web-skimming breaches is all about the placement of the skimming code so that it can reach valuable payment information. [In the case of British Airways, attackers very tactically picked a single JavaScript resource on the server to intercept both mobile and website payments](https://www.riskiq.com/blog/labs/magecart-british-airways-breach/). This placement not only enabled efficient web skimming but also allowed the skimmer to hide in plain sight because it integrated with both the appearance and functionality of the site.

In Macy’s case there were two targets for the skimmer:

**The checkout page:** This is the obvious target. When a customer pays for a product using their payment information, the skimmer will grab it. When combined with shipping and billing information, it creates a full package of card and personal information, known in the card-skimming worlds as "fullz."

**The Macy's Wallet page:** Similar to many other online stores, Macy's has the added capability to store payment information, so a customer doesn't have to enter it every time during checkout. Macy's calls this their "Macy's Wallet," which is a web interface in which customers can manage their payment methods. Because of how well-integrated this skimmer was, it could also target users managing their payment methods via this interface.

The ability for attackers to skim the Macy's Wallet page is a momentous development for web skimming. For the longest time, having stored payment information was an effective way of avoiding skimming attacks. The attackers targeting Macy's took this as a challenge and made their skimmer multifaceted. It is not just a skimmer for a checkout process; it is a skimmer for valuable information, wherever it may be.

Macy's mentioned in their notice that mobile wasn't affected, noting that "Customers checking out or interacting with the My Account wallet page on a mobile device or the macys.com mobile application were not involved in this incident."

The Inroad: Another Compromised Resource

For a skimmer to access sensitive data, it must be put into the e-commerce platform in some way. Since RiskIQ has tracked Magecart, we've seen this done in dozens of ways, for instance,  in supply-chain attacks, such as the one that compromised Ticketmaster. In that instance a third-party code provider was compromised that integrated with Ticketmaster and more than 800 other e-commerce sites. In the case of Macy's, the compromise was internal—someone had access to files on the webserver and strategically picked a file that would load onto the checkout and Macy's wallet pages.

The file in question was a minified JavaScript file called **ClientSideErrorLog.js,** which was loaded from the following path:

https://www.macys.com/js/min/common/util/ClientSideErrorLog.js

On a typical day, this script would look like this, small and minified:



The script was modified to include some somewhat obfuscated skimming code (the code continues far outside of this screenshot as the skimmer is quite extensive):



The same script also exists as a non-minified resource on the Macy’s website, located at a similar path (it just is just missing the “min” keyword):  
  
https://www.macys.com/js/common/util/ClientSideErrorLog.js

We never observed this resource to contain the malicious code; it was isolated in the minified script. From this script, the attackers had the checkout page and wallet page in hand. They spent a long time working out their skimmer before placing it here—RiskIQ's timeline suggests this infrastructure behind the attack went up on September 24th, while the skimmer didn't become active until October.

We want to thank an anonymous researcher for sharing a sample of his observations, which allowed us to exclude our suspicion that the skimmer might have been updated over time. Community collaboration is always highly appreciated because none of us are fighting these bad guys alone.

The Skimmer: Malicious Code Custom-Built for the Macy’s Checkout Flow

In all the years that RiskIQ has been researching, analyzing, detecting, and mitigating Magecart attacks, we've never observed a skimmer so customized as the one used to attack Macy's. Unlike the majority of Magecart skimmers, this skimmer could work only for the Macy's website. From analyzing the skimmer, we can tell this isn't the first time these attackers have performed skimming attacks. However, they don't map to any known Magecart Group. We have also yet to see stolen data from this breach go up for sale anywhere as of this writing.

As we mentioned, the skimmer has two targets—the checkout page and the Macy's wallet page—but also two modes of operating on the checkout pages. For the Macy's wallet page, a user has to be logged in, meaning they have a Macy's account. However, to perform a checkout, a visitor does not have to log in; they can do a guest checkout. The skimmer is aware of this and during checkout will go into one of two flows: guest checkout or registered guest checkout.

When the skimmer activates in the visitor's browser it starts with an initialization function in which it determines the following:

1. Is it on the checkout page:
   1. If so, is it a registered user or is it a guest
2. Is it on the wallet page

In both 1 and 2  (and just like any other skimmer), this skimmer hooks into either the "purchase" or "save card to wallet" buttons, so when a visitor finishes checking out, the data is skimmed and stolen. The skimmer does this by checking the URL of the page the user is on at that moment. Here is the deobfuscated initialization of the skimmer:



In the red box is the checkout (be it guest or registered user) setup and in the green box the wallet setup.

Once this setup is finalized, the Skimmer waits for the user to click the button it's hooked into. Once this happens, the specific skimming scenarios fire. We'll break these down in separate subsections.

Guest Checkout Skimming

When a user is **not** logged in, the skimmer hooks three buttons, the three individual steps a guest performs during checkout:

* **Confirming shipping information**:

At this step, when the user confirms they entered the correct shipping information, the skimmer grabs the input field data containing the address, like so:



* **Confirming payment information**:

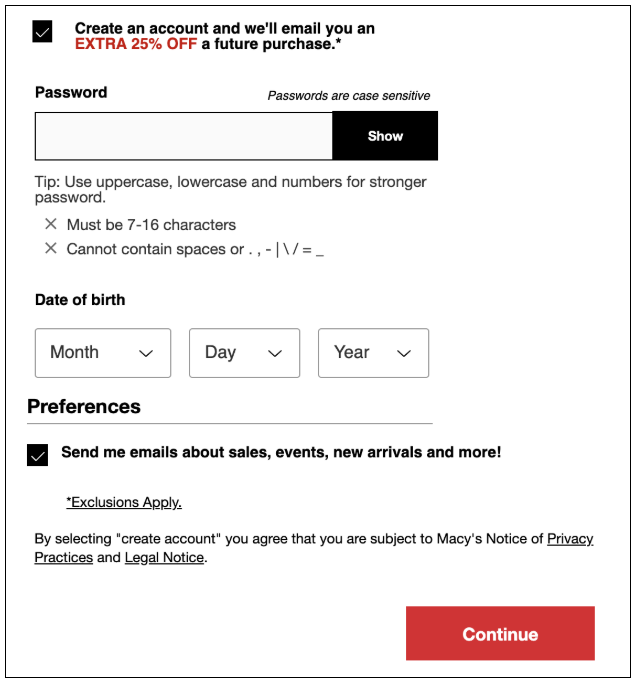
At this step, when the user confirms they entered the correct payment information, the skimmer grabs the input field data containing the payment information, like so:



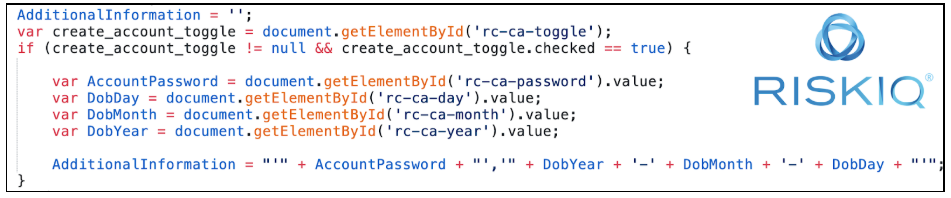
* **Confirming the order**:

At this final step, the user will confirm the order, and the skimmer will check if it was successful in grabbing the shipping and payment information. If so, it will send the stolen information off to the bad guy’s server for collection. We’ll show exactly how this data is sent off in a later subsection.

There is one additional piece of information the attackers manage to skim: login credentials from guests creating accounts during checkout. On the Macy’s checkout page, you can, as a guest, also make an account for your order in case you don’t have one. This creation of an account happens on the checkout form, and they incentive doing so with a 25% off coupon:



Because customers enter this data on the checkout page, the skimmer has access to it. The skimmer will check if someone creates an account. If so, it will skim the newly minted login credentials:

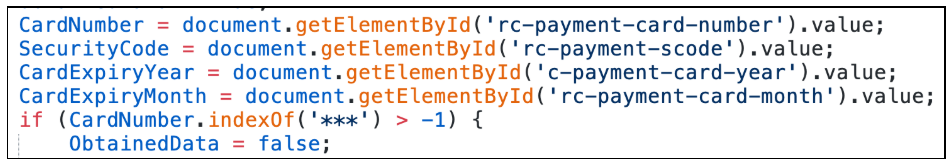


Registered User Checkout Skimming

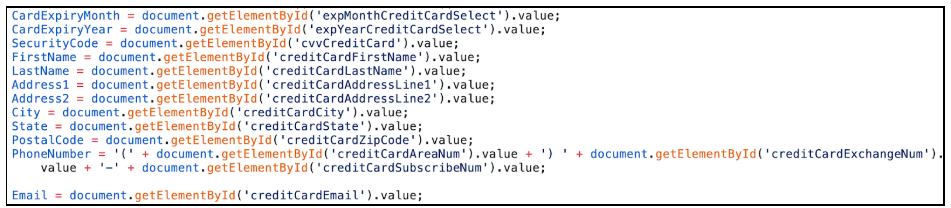
If a registered and logged-in user performs the checkout process, the skimming process becomes slightly different, mainly because the user's payment and shipping information are saved in their account and doesn't need to be filled out. The criminals thought of this and worked out a way to still get to this data.

**Getting the Card data:**

The skimmer will check if the user fills out their card data themselves by skimming it just as it would a guest checkout. It will then check if the credit card number contains at least three asterisk symbols. They run this check because asterisks appear in the form when a user is using a stored payment method. Here is the code for it:



Obfuscated credit card numbers are not always an issue for the Magecart skimmer, which has another method up its sleeve to get the card number. If it sees a masked number, it simply hooks into the add/edit/remove payment data functionality of the page. A user can always modify their card data on the page, and if they do, the skimmer will intercept the new number when it's saving as a payment method. The skimmer has a small helper function which fires off when a user edits, removes or adds a payment method during checkout:



In essence, if payment information is somehow used/modified or available on the checkout page, the skimmer has a way to make sure it gets a copy before the card number is masked again.

**Getting the Shipping Information:**

Just like payment data, a user can save their shipping address in their account, meaning there will now be input fields for the skimmer to skim. However, the shipping address is visible to the user and not masked like the saved payment data is. The only problem for the skimmer is that the shipping address is formatted neatly with HTML on the checkout form, not in raw data fields.

The attackers made a custom parser for the shipping form in case it's prefilled, which parses out HTML elements to get the raw address. Parsing code is something we have never seen Magecart attackers do before at this level. Before it tries to parse out the raw data, it does check to see if a saved shipping address was used. If not, it grabs the raw input fields:



If the customer does use a saved address, the skimmer goes into a long piece of code, which parses the raw HTML of the shipping contact information into individual variables of first name, last name, and individual address pieces.



It is relatively extensive, but it works perfectly. The attackers executed flawlessly in implementing this skimmer. It’s incredibly precise.

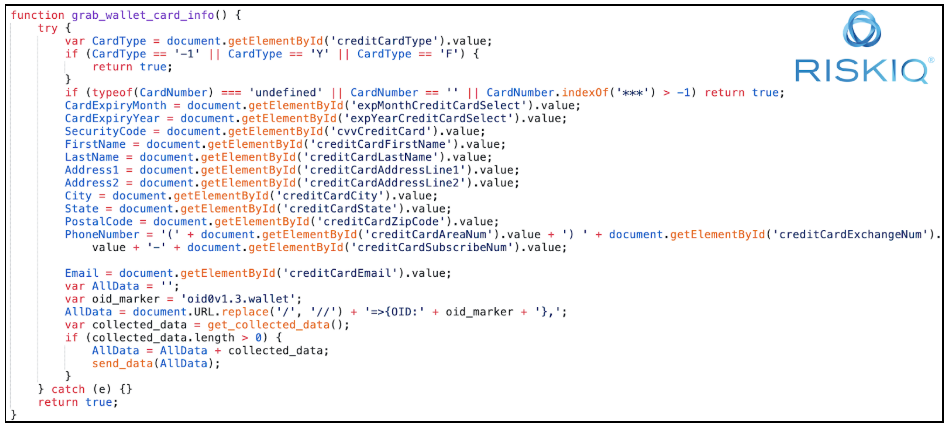
**Sending Off the Stolen Data**:

Once it has obtained all the information it wants—the payment information, most importantly—the skimmer will send the stolen information to the attacker’s server. We’ll discuss how this works in a separate subsection.

Macy’s Wallet Skimming

The unique thing about this skimmer, something we've never observed in any other, is its built-in capability to skim stored payment methods when a user adds or edits them in their account. This ability has traditionally not been a concern for attackers, who have focused solely on the checkout aspect of e-commerce. But as the actors who attacked Macy's know,  this is not the only place payment information lives.

Below, you can see how the skimmer grabs the payment information from the fields as the user enters it. Once a card is saved, the skimmer sends off the stolen data:



It is a straightforward process, but very effective for its simple purpose of stealing more payment information.

Exfiltrating the Stolen Data

The one thing left to explain is how the attackers send off the stolen data and what this looked like during the Macy’s attack.

Once the skimmer has collected the right data, it will send it off to the attacker-owned server. The formatting of the data itself is similar to a JSON format. Here is a breakdown of how it’s formatted (we show it in multi-line format, but it is simply a single-text string):

**<current page url>**=>{OID:**<.guest / .signedin / oid0v1.3.wallet>**},

{CreditCardInfo:**<Skimmed payment and shipping data, separated by commas>**},

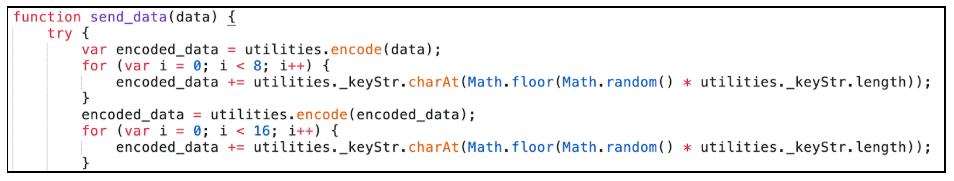
{AdditionalInfo:**<Skimmed account data>**}

Here is an example of what the skimmed data looked like (no real payment data is shown) when being skimmed on the Macy’s wallet page:

https:///www.macys.com/account/wallet?ocwallet=true=>{OID:oid0v1.3.wallet},{CreditCardInfo:Bobby Tables,123456789,123,2019,12,Bobby,Tables,bobby@tables.com,(123)4567890,United States,California,MyTown,Mystreet 123,,12345,macys.com,}

Here, our faux victim, "Bobby Table," had the card number he just added to his Macy's wallet stolen. The tags after the OID marker indicate where the data was skimmed—either from a guest, a logged-in user, or the Macy's wallet page (as in this example). As mentioned, the attackers were fastidious in recording this information.

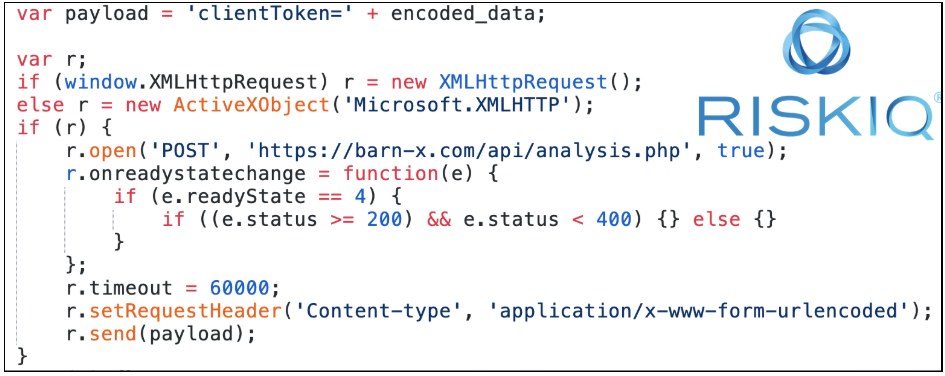
Once this data is concatenated into this data string, it is first encoded with a base64 encoder using a slightly different base64 table. It then adds eight random characters at the end of the encoded data and encodes it with base64 again. It does this another time by adding 16 random characters and again encoding it with base64. These steps are to pad the data so it can't be decoded easily. In the skimmer, it looks like this:



For those dealing with possibly stolen data, this is the base64 index table used by the skimmer:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789=/+

Once this data is encoded it is sent off in a **POST** request with form URL encoding, the parameter clientToken is used as the field name for the data:



The domain used for sending the data, [barn-x.com](https://community.riskiq.com/search/barn-x.com), is a typosquat on a legitimate product conversion service Macy’s uses, which uses the domain bam-x.com. This is a simple attempt to blend in with normal traffic.

What Data Was Stolen, Exactly?

One thing we wanted to make clear was the exact details of what information was stolen when the skimmer did its work. Typically, we would summarize it as “payment data,” but this skimmer was extremely effective in grabbing a wide range of data, beyond just credit cards.

The following data was stolen during checkout/payment data management:

* Full Name
* Address (Street, State, Zipcode, Country)
* Phone Number
* Email address
* Credit Card information (card number and security code)
* Newly created account credentials (Username, password, and date of birth for any account created during checkout)

Conclusion

RiskIQ now detects several Magecart breaches hourly and has observed Magecart skimmers in the wild millions of times. With that kind of volume, it's not surprising that Magecart operatives were working extra hard to get a piece of the billions of dollars consumers were spending on e-commerce platforms during the holiday shopping frenzy. However, while many of these attacks are amateurish, using crude tactics to try to get a skimmer (often bought online pre-packaged) on any site and any webpage possible, the attack on Macy's was different. However, it gives us a good idea of where future attacks are headed.

This Magecart group specifically targeted Macy's. Operatives studied the website closely leading up to the attack, and experienced cybercriminals meticulously learned the ins and outs of the Macy's e-commerce platform. The result was a skimmer custom-built to exfiltrate as much Macy's customer data as possible and stay hidden while doing it.

Highly targeted, highly technical breaches may become a trend. We learned from the Macy's breach that there are a variety of ways to attack the functionality of a website, and operatives with the right acumen and enough time will find them. In this case, the attackers unlocked the ability to skim saved payment information from customers, a capability rarely seen in the wild before this attack.

Unfortunately, given the lucrative nature of card skimming, Magecart attacks will continue to evolve and surprise security researchers with new capabilities. They're learning from past attacks to stay one step ahead, so it's on us to do the same. [Make sure you're staying up to date by reading all our findings on Magecart](https://www.riskiq.com/blog/category/magecart/) and stay tuned as we continue to shine a light on new developments. Also, find out how RiskIQ protects customers [by reading up on our JavaScript Threats Module here](https://www.riskiq.com/blog/external-threat-management/javascript-threats/).