TrickBot Bolsters Layered Defenses to Prevent Injection Research

Michael Gal ⋮

This post was written with contributions from IBM X-Force's Limor Kessem and Charlotte Hammond.

The cyber crime gang that operates the TrickBot Trojan, as well as other malware and ransomware attacks, has been escalating activity. As part of that escalation, malware injections have been fitted with added protection to keep researchers out and get through security controls. In most cases, these extra protections have been applied to injections used in the process of online banking fraud — TrickBot's main activity since its inception after the Dyre Trojan's demise.

IBM Trusteer researchers analyzed TrickBot's most recent injections and the anti-analysis techniques used to conceal their activity. The information is provided in detail in this post.

MiTB — The Basics

Man-in-the-browser (MiTB) attacks are a way for adversaries to intercept the communication between users or users and remote services. The most common use of this interception is by banking Trojans during web sessions. MiTB scripts are designed to modify information going out of the browser on the fly so that what reaches the bank's server fits the criminal's directions.

TrickBot is one of the most modular and sophisticated modern Trojans. It uses a variety of injections, some of which are very advanced, to trick both users and their service providers in order to commit bank fraud. In TrickBot's case, injections can either be fetched locally from configuration files or in real-time from the attacker's inject server.

While one might be able to extract a list of TrickBot targets from its configuration files, things get a lot harder for those seeking to understand what activity will be launched against each target. As with other banking Trojans, the attack tactics change for each bank to match the challenges fraudsters will encounter before a transaction is authorized.

First Line of Defense: Server-Side Injection Delivery

Keeping injections on infected machines means they are more likely to land in the hands of security researchers. Injections kept locally are also less agile and harder to manipulate in real-time. To move beyond these risks, TrickBot's operators inject from their server, known as server-side injections. To facilitate fetching the right injection at the right moment, the resident TrickBot malware uses a downloader or a JavaScript (JS) loader to communicate with its inject server.

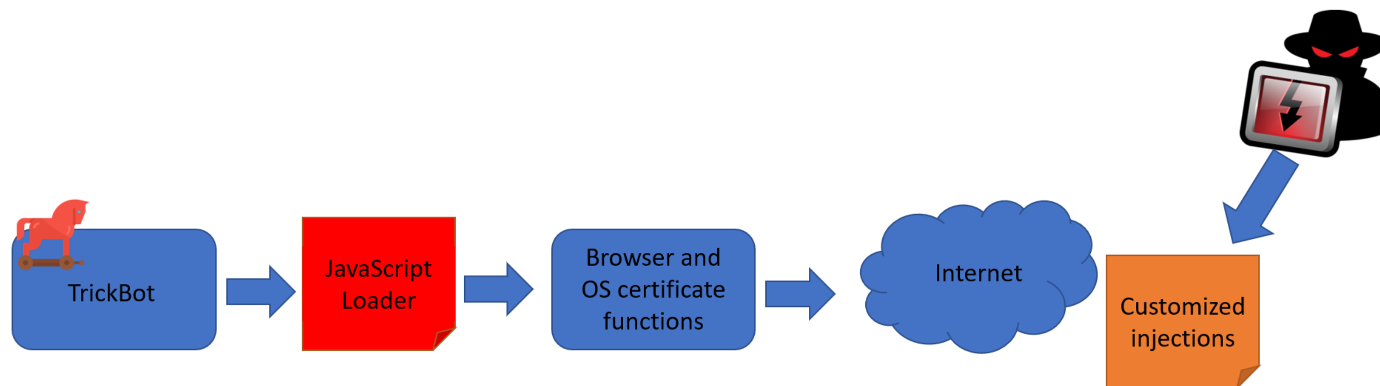

Figure 1: TrickBot's server-side injection flow

Second Line of Defense: Secure Communications With the C2

For operational security, the JS downloader fetches injections via a secure request using the HTTPS protocol to the attacker's command and control (C2) server. It provides that request using a referrer policy parameter with the flag 'unsafe-URL.' The flag specifies that a complete URL, stripped for use as a referrer, is to be sent along with both cross-origin requests and same-origin requests made from a particular client. In TrickBot's case, using this flag likely provides information about the specific page the user is browsing to the C2 server, allowing the C2 server to send custom injections per page. The attacker can also use the information to ignore requests from unwelcome/unknown sources or pages, thus sending nothing back to the client side.

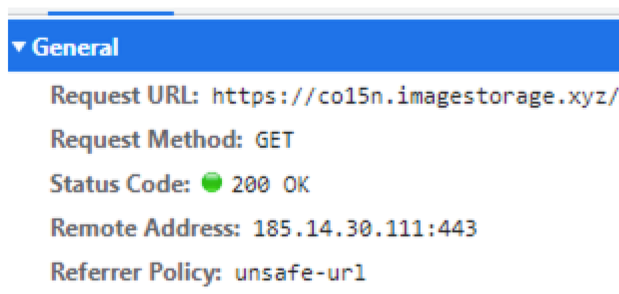

Figure 2: HTTPS requests to the C2 server with referrer policy

For secure communication with a remote inject server, TrickBot hooks the certificate verification function on the infected device. It thereby blocks any certificate errors that the victim might otherwise be alerted to during the malicious communication with the attack server.

The request to the C2 server yields a web injection that was designated by the attacker for each targeted bank URL. Each injection is used to interact with the victims and trick them into divulging details that will help the attack finalize the transaction.

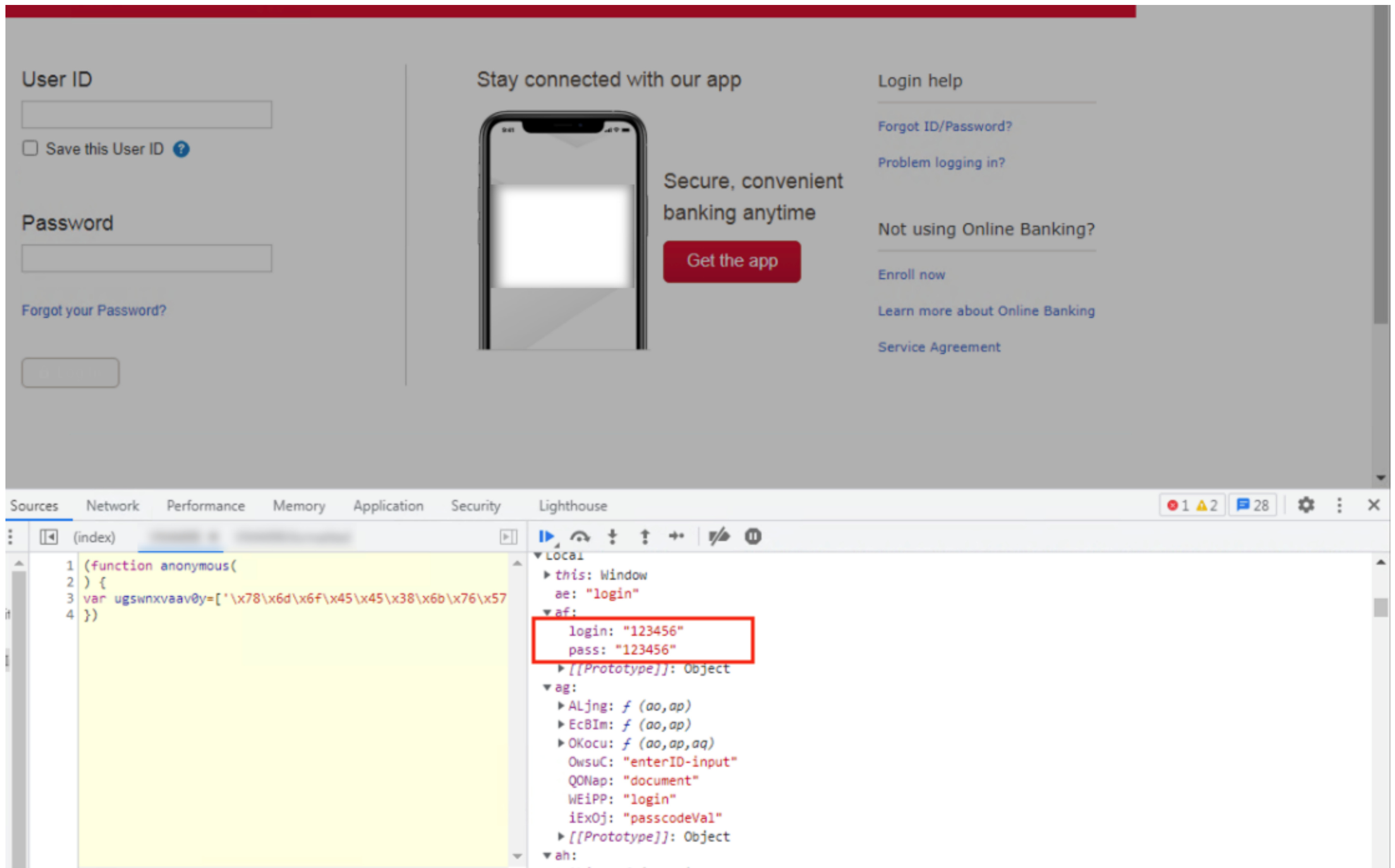

Figure 3: Web injections collecting user credentials on an infected device

A glance at the data collected also shows the injection scrapes the device's fingerprint and sends it to the C2 server.

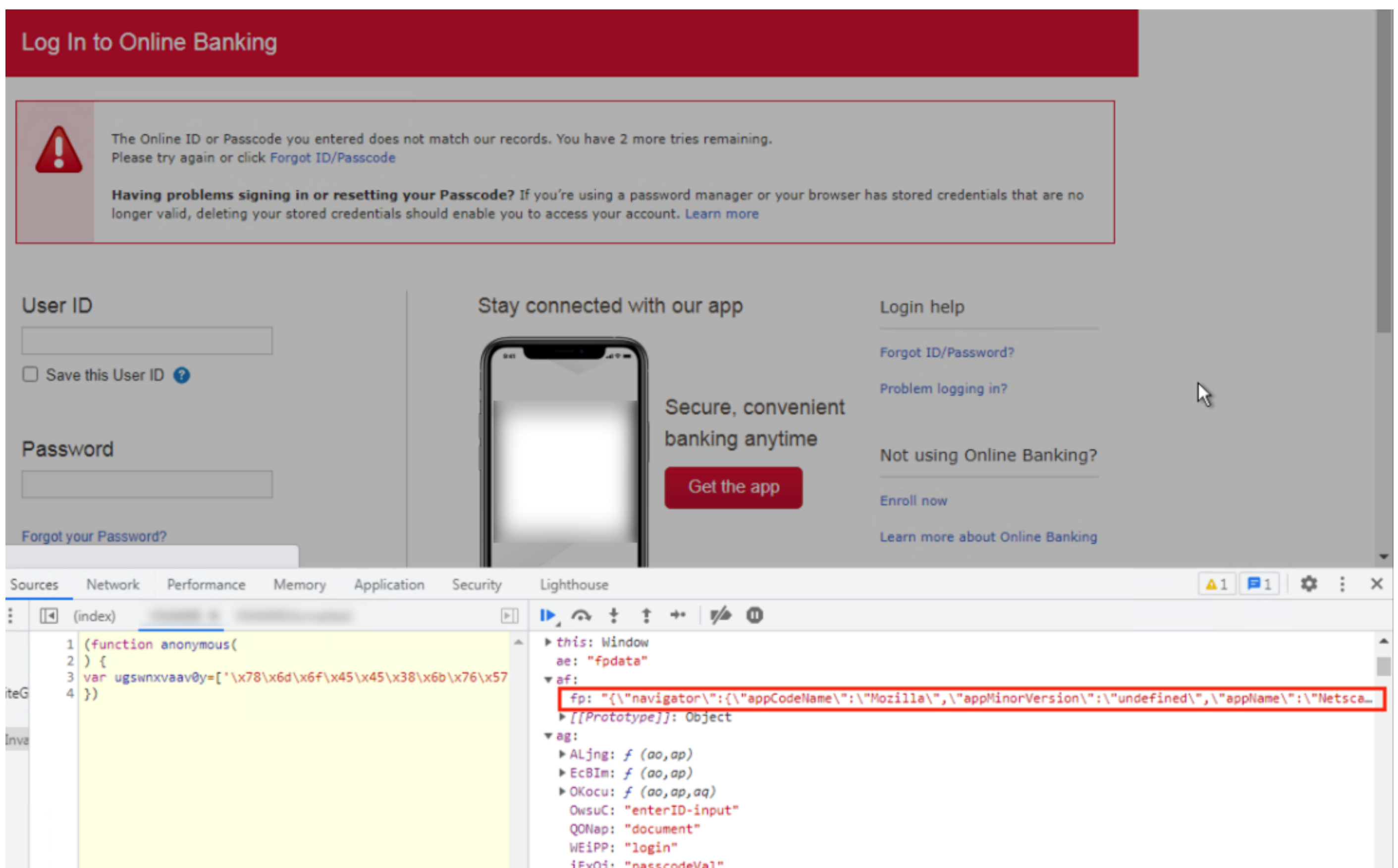

Figure 4: Web injections collecting fingerprint on an infected device

The fingerprint data is quite elaborate, including browser parameters, language settings, operating system basics, user agent, plugins and more. Stolen fingerprint data helps attackers get more information about each session and impersonate victims in fraudulent activity.

Third Line of Defense: Anti-Debugging

To further protect its injections, TrickBot added an anti-debugging script to the JS code. The goal is to anticipate the typical actions researchers will take and ensure their analysis fails. In this case, TrickBot can trigger a memory overload that would crash the page and hinder the analysis.

One of the ways TrickBot's developer achieved this is by looking for 'code beautifying' performed by the researcher. When someone encounters a large block of code that is very 'messy' to a human eye, they will apply 'beautifying' to it. For instance, when looking at obfuscated injection code, a researcher may start by decoding it from the Base64 format, then make all literals and functions human readable. Literal values are changed to real ones, code is divided into chunks, etc. All these efforts are part of code beautifying, and TrickBot expects that from researchers, making it a good place to hold them back.

Searching for code beautifying, TrickBot uses a RegEx to check a function named 'this['Ccmdra'].' If this function is run through the eval(atob function in its original state, it will appear as this['Ccmdra'] = function(){return'newState';} without any new lines or spaces, which are typically added when someone beautifies code.

However, if the code was beautified by someone, they would likely remove the base64 encoding and replace the function with its decoded contents. It might look more orderly.

this['Ccmdra'] = function(){

return'newState';

}

```
        this['firstLineREGEX'] = '\x5cw+\x20*\x5c(\x5c)\x20*{\x5cw+\x20*',// \\w+ *\\(\\) *{\\w+ *
        this['secondLineREGEX'] = '[\x27|\x22].+[\x27|\x22];?\x20*}';//|"].+['|"];? *
    };
    _0x471476['prototype']['isTheCodeBeautified'] = function() {
        var _0xe8f47 = new RegExp(this['firstLineREGEX'] + this['secondLineREGEX'])
          , _0x2b12b3 = _0xe8f47['test'](this['Ccmdra']['toString']()) ? --this['fkIGaK'][-0x1 * -0x2045 + -0x24df + 0x9 * 0x83] : --this['fkIGaK'][0x47e + -0x1ca2 + 0x3c * 0x67];
        return this['BjjjOo'](_0x2b12b3);
    }
    ,
    _0x471476['prototype']['BjjjOo'] = function(_0x38c1e4) {
        if (!Boolean(~_0x38c1e4))
            return _0x38c1e4;
        return this['BmBmxG'](this['slmZBo']);
    }
    ,_0x471476['prototype']['BmBmxG'] = function(_0x28e57e) {
        for (var _0x382585 = 0, _0x5c31b7 = this['fkIGaK']['length']; _0x382585 < _0x5c31b7; _0x382585++) {
            this['fkIGaK']['push'](Math['round'](Math['random']())),
            _0x5c31b7 = this['fkIGaK']['length'];
        }
        return _0x28e57e(this['fkIGaK'][0]);
    }
```

Figure 5: Is the code beautified?

```
decoyFunction = function(){return'newState';}

function infinityLoop(){
    dynamicArray = [1, 0, 0];
    increasingNumber = dynamicArray.length;
    for (var i = 0; i < increasingNumber; i++) {
            dynamicArray.push(Math.round(Math.random()));
            increasingNumber = dynamicArray.length;
        }
}

function isTheCodeBeautified(){
    var regexCheck = new RegExp("\\w+ *\\(\\) *{\\w+ *['|\"].+['|\"];? *}");
    if (regexCheck.test(decoyFunction.toString()))
        keepRunning();
    else
        infinityLoop();
}
```

Figure 6: Now the code is beautified

TrickBot uses a RegEx to detect the beautified setup and throw itself into a loop that increases the dynamic array size on every iteration. After a few rounds, memory is eventually overloaded, and the browser crashes.

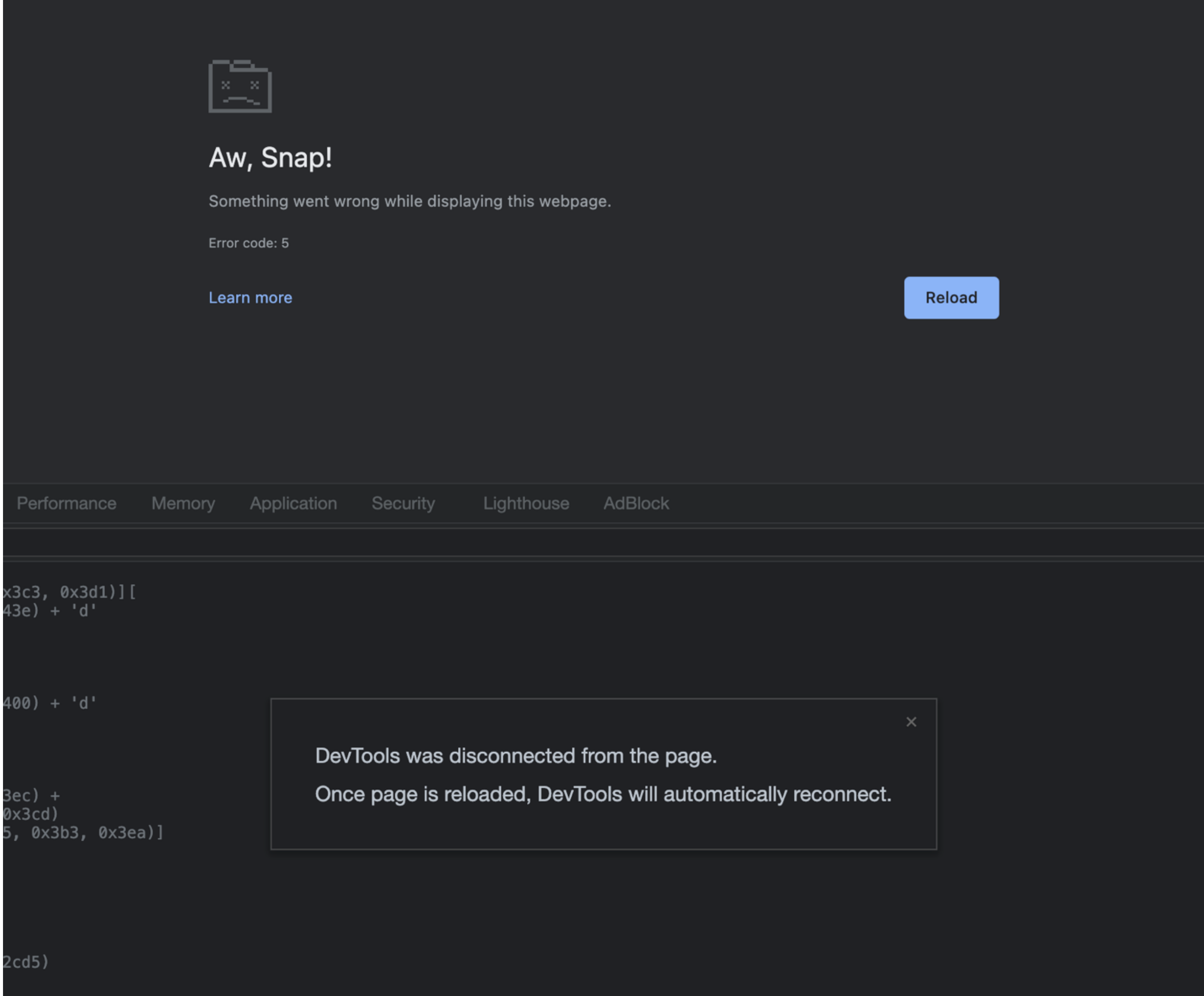

Figure 7: Browser crashes after memory overload

Fourth Line of Defense: Obfuscation and Encoding

The code TrickBot injects is meant to be obfuscated. It is first encoded with Base64 so that scripts are not in plain text. The following techniques are used by TrickBot's developers to achieve some added obfuscation:

Minify/Uglify — Through a series of transformations, such as variables, function, arguments renaming and string removal, TrickBot's code is made to appear unreadable to human eyes, all while working the same.

String extraction and replacement — window.console.log(1) can be written as window['console']['log'](1). The obfuscator moves all strings to an array and encrypts them, which hides functions, literals, arguments and information about the program's execution.

Number base and representing — TrickBot uses hex representation to represent numbers and initialize variables into some value in a complex way. For example, instead of writing:

var num = 0;

This could be replaced with:

write var num = (0x130 * 0x11 + -0x17f5 + 0x2e * 0x15, -0x24a5 + 0x68e * -0x4 + 0x3edd, -0x17f1 + -0x99b * 0x3 + 0x34c2)

This tactic is common to many obfuscators.

Dead code injection — To make things more confusing, TrickBot's developers added redundant code. This makes code less readable and renders its actions harder to decipher. This resembles malware that imports redundant modules to hide their true purpose.

Monkey patching — Patching native functions to change their behavior in a way that makes it impossible to understand what is being activated using static analysis.

Dominating the Cyber Crime Arena

The TrickBot Trojan and the gang that operates it have been a cyber crime staple since they took over when a predecessor, Dyre, went bust in 2016. TrickBot has not rested a day. Between takedown attempts and a global pandemic, it has been diversifying its monetization models and growing stronger.

Whether through ransomware attacks or by partnering with other cyber crime gangs and service providers from Eastern Europe, TrickBot remains a concern to businesses on a few fronts. TrickBot goes after corporate money, facilitates ransomware and extortion attacks, and deploys other unwelcome malware on the network.

TrickBot distributes multi-stage malware through phishing emails, malspam, botnets, hijacked email conversations and even a malicious call center known as BazarCall. While themes vary, often a booby-trapped attachment is a way to infect users. The method used in each campaign is shuffled often, which

can make it harder to anticipate. TrickBot incorporates vulnerabilities, lateral movement tools like Cobalt Strike and living-off-the-land tactics like PowerShell scripts.

To mitigate the risk from TrickBot infections, employees should be kept up to date on recent attack tactics, and relevant threat intelligence should be incorporated into the choice of security controls that protect your organization. In addition, you should:

- Use an email security solution to scan, filter and strip attachments as needed. This is especially important for macro-enabled attachments.
- Consider the use of email spoofing prevention protocols to help lower the risk of receiving email from suspicious sources.
- Follow least privilege principles and minimize the number of privileged accounts on networks and cloud assets.
- Activate multi-factor authentication on privileged accounts, and preferably on all accounts.
- Design architectural controls for network segregation. This can help limit lateral movement and minimize broader infection and data theft.
- Monitor for lateral movement.
- Have offline backups and a backup schedule. TrickBot infections are very likely to become ransomware attacks.
- Make sure data classified as confidential or data that could expose customers and the organization to extortion is encrypted (on-prem and in the cloud). Ransomware attacks often include data extortion.
- Continue role-based employee training to ensure the organization's employees protect the organization from malware.
- Learn about zero trust and begin your zero trust journey, if you haven't done so already.

For more about IBM Trusteer, visit: www.ibm.com/security/fraud-protection/trusteer

IOCs From IBM's Analysis

Domains/ resources

hxxps:\/\/myca.adprimblox.fun

hxxps:\/\/ksx.global-management-holdings.com

hxxps:\/\/on.imagestorage.xyz

hxxps:\/\/997.99722.com

hxxps:\/\/akama.pocanomics.com

hxxps:\/\/web7.albertleo.com

IP addresses

- 94.242.58.165
- 185.14.30.111
- 208.115.238.183

- 51.83.210.212
- 103.119.112.188
- 185.198.59.85

SHA1 hashes

- jquery-1.10.1.js: 5acd3cddcc921bca18c36a1cb4e16624d0355de8
- downloader js: ae1b927361e8061026c3eb8ad461b207522633f2