# **Assignment3 SEC-Lab**

**Sumaya Altamimi & Nouf Almutawa**

## **Examining Ursnif Infections**

**Ursnif** malware, also known as Gozi/ IFSB, is one of the most widely spread banking Trojan malware. That is effectively delivered through malicious spam campaigns. This spam attachment is a Microsoft office document that instructs the user to enable macro. The Ursnif family of malware has been active for years, and current samples generate distinct traffic patterns.

This report shows reviews of packet captures (pcaps) of infection Ursnif traffic using Wireshark. Understanding these traffic patterns can be critical for security professionals when detecting and investigating Ursnif infections.

This report covers the following:

* Ursnif distribution methods
* Categories of Ursnif traffic
* Five examples of pcaps from Ursnif infections

After executing the instructions of the tutorial in this link, <https://unit42.paloaltonetworks.com/wireshark-tutorial-examining-ursnif-infections/>, the following sections shows the steps and screenshots.

Ursnif Distribution Methods

Logo, company name

Description automatically generated

## **Categories of Ursnif Traffic**

This report covers two categories of Ursnif infection traffic:

* Ursnif **without HTTPS** post-infection traffic
* Ursnif **with HTTPS** post-infection traffic

Malware samples from either of these categories create the same type of artifacts **on an infected Windows host.** For example, both types of Ursnif remain persistent on a Windows host by updating the Windows registry, such as the example shown in Figure 2.

Graphical user interface, text, application, email

Description automatically generated

## **Example 1: Ursnif without HTTPS**

Open the pcap in Wireshark and filter on ***http.request***.

Graphical user interface, application, table, Excel

Description automatically generated

In this example, the Ursnif-infected host generates post-infection traffic to 8.208.24[.]139 (Destination) using various domain names ending with .at.

This category of Ursnif causes the following **traffic**:

* HTTP GET requests **caused by the initial Ursnif binary**
* HTTP GET request **for follow-up data**, with the URL ending in .dat
* HTTP GET and POST requests **after Ursnif is persistent in the Windows registry**

The following **HTTP data** is used during the traffic in our first example:

* Domain for initial GET requests: **w8.wensa[.]at**
* Request for follow-up data: **hxxp://api2.casys[.]at/jvassets/xI/t64.dat**
* Domain for GET and POST requests after Ursnif is persistent: **h1.wensa[.]at**

Follow the TCP stream for the very first HTTP GET request at 20:13:09 UTC.

Graphical user interface, text

Description automatically generated

The TCP stream window shows the full URL. Note how the GET request starts with /api1/ and is followed by a long string of alpha-numeric characters with backslashes and underscores.

Text

Description automatically generated

We can find the same pattern from Ursnif activity caused by a Hancitor infection on December 10,2019.

Text

Description automatically generated with medium confidence

Mixed with the other malware activity, on December 10th example contains the following indicators for Ursnif:

* Domain for initial GET requests: foo.fulldin[.]at
* Request for follow-up data: hxxp://one.ahah100[.]at/jvassets/o1/s64.dat
* Domain for GET and POST requests after Ursnif is persistent: api.ahah100[.]at

Note how patterns from Ursnif traffic in the December 10th example are similar to the patterns we find in example 1. These patterns are commonly seen from Ursnif samples that do not use HTTPS traffic.

## **Example 2: Ursnif with HTTPS**

Like our first pcap, this one has also been stripped of any traffic not related to the Ursnif infection.

Open the pcap in Wireshark and filter on ***http.request or ssl.handshake.type == 1***. For Wireshark 3.0 or newer, filter on ***http.request or tls.handshake.type == 1*** for the correct results.

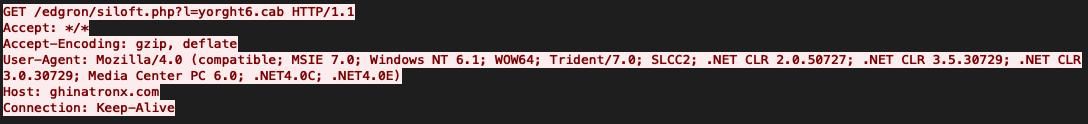
Graphical user interface, table, Excel

Description automatically generated

This example has the following sequence of events:

* HTTP GET request that returns an initial Ursnif binary
* HTTP GET requests caused by the initial Ursnif binary
* HTTPS traffic after Ursnif is persistent in the Windows registry

Follow the TCP stream for the first HTTP GET request to ghinatronx[.]com.



Graphical user interface, text

Description automatically generated

This TCP stream reveals a Windows executable or DLL file. As shown, MZ are two bytes of Windows EXE or DLL represented in ASCII format.

The next four HTTP requests to bjanicki[.]com were caused by the Ursnif binary. Follow the TCP stream for the first HTTP GET request to bjanicki[.]com at 18:46:21 UTC.

Text

Description automatically generated

This TCP stream shows the full URL. Note how the GET request starts with /images/ and is followed by a long string of alpha-numeric characters with backslashes and underscores before ending with .avi. This URL pattern is somewhat similar to Ursnif traffic from our first example. Unlike our first example, Ursnif in this second pcap generates **HTTPS** traffic after it becomes persistent on an infected Windows host.

Use your **basic** web filter for a quick review of the HTTPS traffic. Note the HTTPS traffic to prodrigo29lbkf20[.]com.

Graphical user interface

Description automatically generated

As shown, these are the http traffic caused by Ursnif.

HTTPS traffic generated by this Ursnif variant reveals distinct characteristics in certificates used to establish encrypted communications.

To get a closer look, filter on***ssl.handshake.type == 11*** (or ***tls.handshake.type == 11*** in Wireshark 3.0 or newer).

Table

Description automatically generated

Select the first frame in the results and go to the frame details window. There we can expand lines and work our way to the certificate issuer data.

Text

Description automatically generated

As shown, we expand the line for ***nsport Layer Security***in the frame details window. Then we expand the line labeled ***TLSv1.2 Record Layer: Handshake Protocol: Certificate***. Then we expand the line labeled ***Handshake Protocol: Certificate***. We keep expanding, until we find our way to the certificate issuer data as shown:

Text

Description automatically generated

Individual items under the **rdnSequence** line show properties of the certificate issuer. These reveal the following characteristics:

* countryName=**XX**
* stateOrProvinceName=**1**
* localityName=**1**
* organizationName=**1**
* organizationalUnitName=**1**
* commonName=**\***

This issuer data is not valid, and these patterns are commonly seen in Ursnif infections.

But what does legitimate certificate data look like? Below is a valid data from a certificate issued by DigiCert.

* countryName=**US**
* organizationName=**DigiCert Inc**
* commonName= **DifiCert SHA2 Secure Server CA**

One last thing about Ursnif is the IP address check by an Ursnif-infected host. This happens over DNS using a resolver at opendns[.]com. Like other IP address identifiers, this is a legitimate service. However, these services are commonly used by malware.

To see this IP address check, filter on ***dns.qry.name contains opendns.com*** and review the results.

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

This is the IP address of the infected Windows host.

As shown in the Figure:

* The Window host generated a dns query for resolver1.opendns[.]com
* This is followed by a DNS query to 208.67.222[.]222 for myip.opendns[.]com.
* The DNS query to myip.opendns[.]com returned the public IP address of the infected Windows host.

## **Example 3: Ursnif with Follow-up Malware**

This pcap also has unrelated activity stripped from the traffic, but it builds on our last example. Our third pcap includes what appears to be decoy traffic, and it also includes an HTTP GET request for follow-up malware.

The sequence of events is:

* HTTP GET request that **returns** an initial Ursnif binary
* HTTP GET requests **caused by** the initial Ursnif binary, including decoy URLs
* HTTPS traffic after Ursnif is **persistent** in the Windows registry
* HTTP GET request for **follow-up** malware

Use your basic web filter for a quick review of the web-based traffic.

Graphical user interface, table

Description automatically generated

In Figure, the initial HTTP request to sinicaleer[.]com **returned** a Windows executable for Ursnif. The remaining traffic was **caused by** the Ursnif executable until it became **persistent**.

* Three HTTP requests to google[.]com follow similar URL patterns as Ursnif traffic to an actual malicious domain of ghdy656262oe[.]com.
* These HTTP GET requests to google[.]com appear to be decoy traffic, because they do not assist the infection.
* HTTPS traffic over TCP port 443 to gmail[.]com and www.google[.]com also serves no direct purpose for the infection, and that activity could also be classified as decoy traffic.

The Figure shows an example of the **decoy HTTP GET requests** to google[.]com :

Text

Description automatically generated

Note the HTTP traffic to ghdy656262oe[.]com.

The first two GET requests to ghdy656262oe[.]com return a ***404 Not Found*** response as shown:

Graphical user interface, text, application

Description automatically generated

The third HTTP GET request returns a ***200 OK*** response, and the infection continues as shown:

Graphical user interface, text

Description automatically generated

Since the first HTTP GET request to ghdy656262oe[.]com was not a 200 OK, the infected Windows host cycled through other malicious domains to continue the infection. These two domains are tnzf3380au[.]top and xijamaalj[.]com.

However, the DNS queries for these domains returned a “No such name” in response, so the infected Windows host went back to trying ghdy656262oe[.]com.

Use the following Wireshark filter to better see this sequence of events: ***((http.request or http.response) and ip.addr eq 194.1.236.191) or dns.qry.name contains tnzf3380au or dns.qry.name contains xijamaalj***

Table

Description automatically generated with medium confidence

To review the rest of the infection, use your ***basic*** web filter and scroll to the end of the results. The Figure shows the post-infection traffic after Ursnif becomes **persistent**:

Text

Description automatically generated

After five HTTP GET requests to ghdy656262oe[.]com, we find traffic generated by the infected Windows host after Ursnif becomes persistent. This includes HTTPS traffic to google[.]com and gmail[.]com.

Table

Description automatically generated with medium confidence

Traffic to vnt69tnjacynthe[.]com should have the same type of certificate issuer data we witnessed in our second pcap.

But this traffic includes an HTTP GET request to carresqautomotive[.]com ending with .rar.This URL ending in .rar returned follow-up malware. However, this follow-up malware is encoded or otherwise encrypted when sent over the network. The binary decoded on the infected Windows host, which is not seen in the infection traffic.

Follow the TCP stream for the HTTP GET request to carresqautomotive[.]com.

Graphical user interface, text, application

Description automatically generated

It shows that it is compressed, but the binary is encoded or encrypted data and not a rar archive, as shown below:

Timeline

Description automatically generated with medium confidence

This data is encrypted, so we cannot export a copy of the follow-up malware from the pcap. Therefore, we must **rely on other post-infection traffic** to determine what type of malware was sent to the Ursnif-infected host. We have seen various types of follow-up malware from Ursnif infections, including [Dridex](https://www.malware-traffic-analysis.net/2019/12/02/index.html), [IcedID](https://www.malware-traffic-analysis.net/2019/10/21/index.html), [Nymain](https://www.malware-traffic-analysis.net/2019/05/03/index.html), [Pushdo](https://www.malware-traffic-analysis.net/2019/07/29/index.html), and [Trickbot](https://www.malware-traffic-analysis.net/2019/09/04/index.html). Our next example is an Ursnif infection with Dridex as the follow-up malware.

## **Example 4: Ursnif Infection with Dridex**

Unlike our first three examples, this pcap example does not have unrelated activity stripped from the traffic. Use your ***basic*** web filter to get a better idea of the traffic.

Graphical user interface, application, table, Excel

Description automatically generated

This pcap has the same sequence of events as our previous example, but **it adds post-infection activity** from the follow-up malware:

* HTTP GET request that **returns** an initial Ursnif binary
* HTTP GET requests **caused by** the initial Ursnif binary, including decoy URLs
* HTTPS traffic **after Ursnif is persistent** in the Windows registry
* HTTP GET request **for follow-up malware**
* **Post-infection activity** from the follow-up malware

In this fourth example, the HTTP GET request for an initial Ursnif binary is to oklogallem[.]com.

Graphical user interface, application, table, Excel

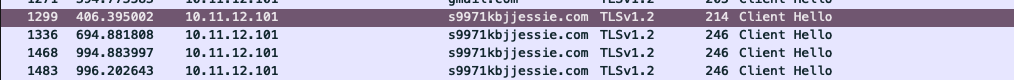
Description automatically generated

Ursnif causes HTTP GET requests to kh2714ldb[.]com before the infection becomes persistent.

Graphical user interface, application, table, Excel

Description automatically generated

Ursnif causes HTTPS traffic to s9971kbjjessie[.]com, as the activity after Ursnif is persistent.



We then see an HTTP GET request to startuptshirt[.]my for the follow-up malware. And the post-infection traffic caused by the follow-up malware.

Text

Description automatically generated with medium confidence

Our fourth example follows the same infection patterns as our third pcap, but now we also have HTTPS/SSL/TLS traffic to 94.140.114[.]6 and 5.61.34[.]51 without any associated domain name. This is **Dridex post-infection traffic**. Certificate issuer data for **Dridex** is different than certificate issuer data for **Ursnif**.

Use the following filter to review the **Dridex certificate data** in our fourth pcap: ***(ip.addr eq 94.140.114.6 or ip.addr eq 5.61.34.51) and tls.handshake.type eq 11***

Graphical user interface, text, application

Description automatically generated

Select the first frame in the results, go to the frame details window, and expand the certificate-related lines.

Text

Description automatically generated

Under the ***rdnSequence*** line, we find properties of the certificate issuer.

Certificate issuer characteristics for HTTPS/SSL/TLS traffic at 94.140.114[.]6 follows:

* countryName=**NP**
* localityName=**Kathmandu**
* organizationName=**Buvecoww Fftaites O.V.E.E.**
* organizationalUnitName=**Olfo Dusar Latha**
* commonName=**ndltman-dsamutb.spiegel**

Certificate issuer data is different for 5.61.34[.]51:

Text

Description automatically generated

But it follows a similar style:

* countryName=**MU**
* localityName=**Port Louis**
* organizationName=**Ppoffi Sourinop Cooperative**
* organizationalUnitName=**ipeepstha and thicioi**
* commonName=**plledsaprell.Byargt9wailen.voting**

This type of issuer data is commonly seen for **Dridex post-infection traffic**. In our next example, we can further practice reviewing certificate issuer data for **Dridex**.

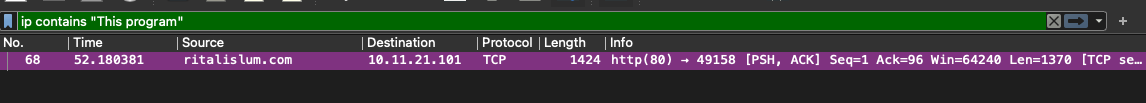
## **Example 5: Evaluation**

Like our previous example, this pcap example has an **Ursnif infection followed by Dridex**, so we can practice the skills described so far in this tutorial.

Based on what we have learned so far, open the fifth pcap in Wireshark, and answer the following questions:

For the initial Ursnif binary, which URL returned a Windows executable file?

* The only Windows executable file in this pcap is the initial Windows executable file for Ursnif. We can use the following Wireshark search filter to quickly find this executable: ***ip contains "This program"***



This filter should provide only one frame in the results. Follow the TCP stream for this frame.

* This is the URL info for this HTTP request

Text

Description automatically generated

* This is an indicate that the URL returned a Windows executable file.



* So, the URL is hxxp://**ritalislum[.]com**/*obedle/zarref.php?l=sopopf8.cab*

After the initial Ursnif binary was sent, the infected Windows host contacted different domains for the HTTP GET requests. Which domain was the traffic successful and allowed the infection to proceed?

By using the **basic** web filter for an overview of the web traffic to find HTTP requests caused by this variant of Ursnif that start with GET /images/ the result as follow:

The first HTTP request to k55gaisi[.]com  returns a 404 Not Found as the response.

Also, the next HTTP GET request for an Ursnif-style URL is to bon11ljgarry[.]com and other request to leinwqoa[.]com, both reveals a redirect to a URL at www.search-error[.]com.

But all the other requests for k55gaisi[.]com as shown below returns 200 ok.

Text

Description automatically generated

* the domain is k55gaisi[.]com. So, from this point, the Ursnif infection proceeds, and we find no further Ursnif-style HTTP requests that start with GET /images/.

What domain was used in HTTPS traffic **after Ursnif became persistent** on the infected Windows host?

When Ursnif is persistent, we no longer see Ursnif-style HTTP requests starting with GET /images/. Instead, we find Ursnif-related HTTPS traffic. Shortly after the final Ursnif-style HTTP GET request, HTTPS traffic to n9maryjanef[.]com begins as highlighted in the Figure.

Table

Description automatically generated

* This is Ursnif traffic. The domain is n9maryjanef[.]com.

We can confirm this is Ursnif traffic by filtering on ***ip.addr eq 185.118.165.109 and ssl.handshake.type == 11.***

Graphical user interface, text

Description automatically generated

By reviewing the certificate issuer data, it looks the same as our second example.

Graphical user interface, text

Description automatically generated

What URL ending in .rar was **used to send follow-up malware** to the infected Windows host?

HTTP GET requests caused by Ursnif for follow-up malware end in .rar, so we can use the following filter to find this URL in our pcap: ***http.request and ip contains .rar***

Graphical user interface, text, application

Description automatically generated

The results are as follow:

Text

Description automatically generated

Text

Description automatically generated

As shown, the HTTP GET request in Figure 30 redirects to an HTTPS URL.

What IP addresses were used for the Dridex post-infection traffic?

One of these IP addresses is the same as Dridex in our fourth pcap, and it has the same certificate issuer data.

185.99.133.38 and 5.61.34.51

Dridex traffic to 185.99.133[.]38 has the same style of certificate issuer data as seen in example 4. Traffic to both IP addresses does not involve a domain name.

The Dridex post-infection traffic is easy to spot in this example if we look for any HTTPS/SSL/TLS traffic without a domain after the HTTP GET request ending in .rar as shown in the Figure below:

Table

Description automatically generated

After the request with the URL that ends with .rar, the upper part is traffic caused by Dridex, while the lower part is HTTPS traffic caused by Ursnif.

## **Conclusion & References**

This tutorial provided tips for examining **Windows infections with Ursnif malware**.

[1] <https://unit42.paloaltonetworks.com/wireshark-tutorial-examining-ursnif-infections/>

[2] [Customizing Wireshark – Changing Your Column Display](https://unit42.paloaltonetworks.com/unit42-customizing-wireshark-changing-column-display/)

[3] [Using Wireshark – Display Filter Expressions](https://unit42.paloaltonetworks.com/using-wireshark-display-filter-expressions/)

[4] [Using Wireshark: Identifying Hosts and Users](https://unit42.paloaltonetworks.com/using-wireshark-identifying-hosts-and-users/)

[5] [Using Wireshark: Exporting Objects from a Pcap](https://unit42.paloaltonetworks.com/using-wireshark-exporting-objects-from-a-pcap/)

[6] [Wireshark Tutorial: Examining Trickbot Infections](https://unit42.paloaltonetworks.com/wireshark-tutorial-examining-trickbot-infections/)