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.

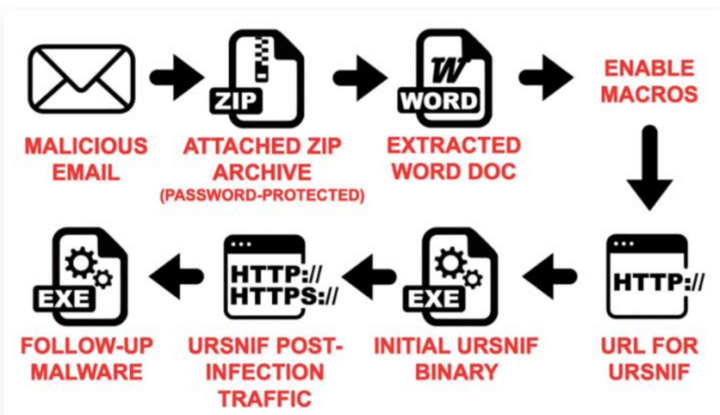
This report shows reviews of packet captures (pcaps) of infection Ursnif traffic using Wireshark. Understanding these traffic patterns can be critical for security professionals when detecting and investigating Ursnif infections.

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

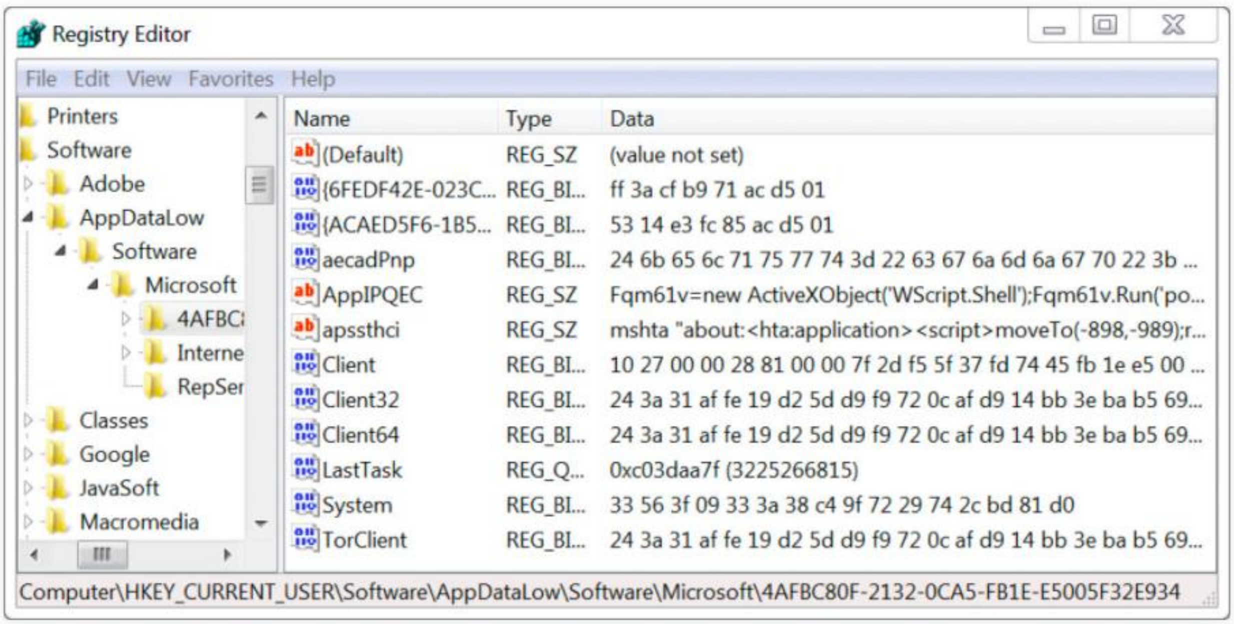


Categories of Ursnif Traffic

This report covers two categories of Ursnif infection traffic:

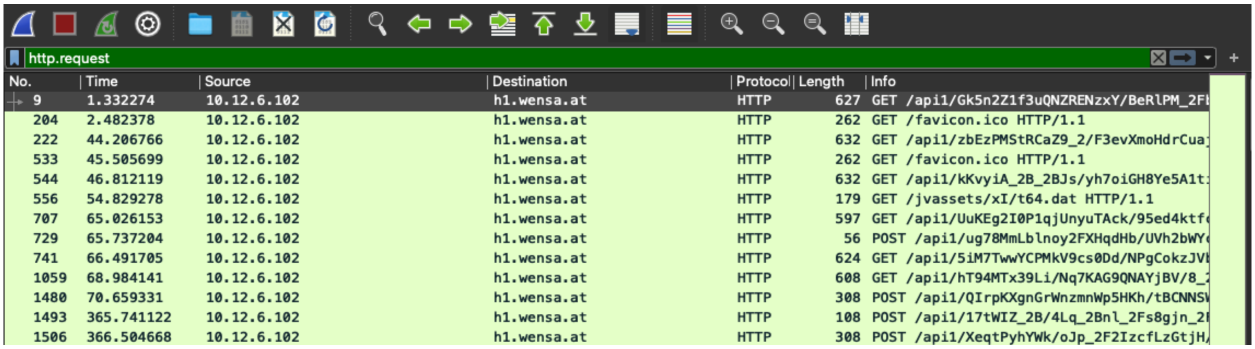
- Ursnif **without HTTPS** post-infection traffic
- Ursnif **with HTTPS** post-infection traffic

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.



Example 1: Ursnif without HTTPS

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



http.request

No.	Time	Source	Destination	Protocol	Length	Info
9	1.332274	10.12.6.102	h1.wensa.at	HTTP	627	GET /api1/Gk5n2Z1f3uQNZRENzxY/BeRlPM_2F
204	2.482378	10.12.6.102	h1.wensa.at	HTTP	262	GET /favicon.ico HTTP/1.1
222	44.206766	10.12.6.102	h1.wensa.at	HTTP	632	GET /api1/zbEzPMStRCaZ9_2/F3evXmoHdrCua
533	45.505699	10.12.6.102	h1.wensa.at	HTTP	262	GET /favicon.ico HTTP/1.1
544	46.812119	10.12.6.102	h1.wensa.at	HTTP	632	GET /api1/kKvyiA_2B_2BJs/yh7oiGH8Ye5A1t
556	54.829278	10.12.6.102	h1.wensa.at	HTTP	179	GET /jvassets/xI/t64.dat HTTP/1.1
707	65.026153	10.12.6.102	h1.wensa.at	HTTP	597	GET /api1/UuKEg2I0P1qjUnyuTAck/95ed4ktf
729	65.737204	10.12.6.102	h1.wensa.at	HTTP	56	POST /api1/ug78MmLblnoy2FXHqdHb/UVh2bWY
741	66.491705	10.12.6.102	h1.wensa.at	HTTP	624	GET /api1/5iM7TwwYCPMkV9cs0Dd/NPgCokzJV
1059	68.984141	10.12.6.102	h1.wensa.at	HTTP	608	GET /api1/hT94MTx39Li/Nq7KAG9QNAYjBV/8_
1480	70.659331	10.12.6.102	h1.wensa.at	HTTP	308	POST /api1/QIrpKXgnGrWnzmnWp5HKh/tBCNNS
1493	365.741122	10.12.6.102	h1.wensa.at	HTTP	108	POST /api1/17tWIZ_2B/4Lq_2Bnl_2Fs8gjn_2
1506	366.504668	10.12.6.102	h1.wensa.at	HTTP	308	POST /api1/XeqtPyhYWk/oJp_2F2IzcfLzGtjH

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

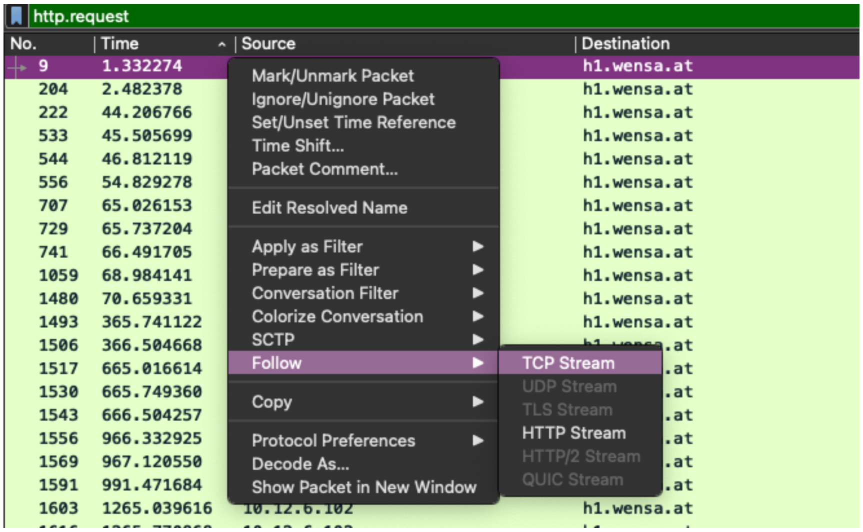
This category of Ursnif causes the following **traffic**:

- 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.



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.

```
GET /api1/Gk5n2Z1f3uQNZRENzxY/BeRlPM_2FbyOTq4aK_2FIp/Dcf3zrOuZ613w/_2Fhq1ZS/yoxosmdDTxp9Df8gT15FbeP/OsS35tjtdS/pG4Ea0Dugz9Ebe3MH/
y5mC9bRgxZqo/fWI3ZuQUVgj/JTT_2BCNRVl47G/Sn4OiLXUssnrQlch1AIfy/fVs5UZordH_2F4JX/VHN3v9rJEXrQciq/FioGSOAvDEVWsPDBsW/hdabxOoBP/
aFTak3JuZQkyahGQ2Dm6/G6fFPHEke56BRc_0A_0/D1H57FB2I6Y_2FZRwi1G8N/C4nUcv3 HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Host: w8.wensa.at
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
DNT: 1
Connection: Keep-Alive
```

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

```
GET /webstore/MXLiOmC0u5dhdKr/NLpVmz4BdejgpgL_2B/G3uY2n_2F/XQTJJZwLO18NDf7hACPi/dQzBldXdJHGynuO_2Fn/ldzAKv61Rh1OdE42cvhZqM/K2m089aA6_2Bw/
eiry019n/rA6D4FF_2BPwIUVPU2l3oKO/No89rIVydY/CAvSD1ufu_2B9H8dv/V5IxTliZ4_2B/B9qyfOul58F/M0e5AERHuRGk4z/JHc89nqzYd0EoJAOtU5wf/
2ctWwmqtpgC9WAUA/HXFbWoC9_2FXKQF/SQEUz_0A_0D_2BhRDC/WCEVMYhO/uassyZGs1/rx_2Fl HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Host: foo.fulldin.at
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
DNT: 1
Connection: Keep-Alive
```

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

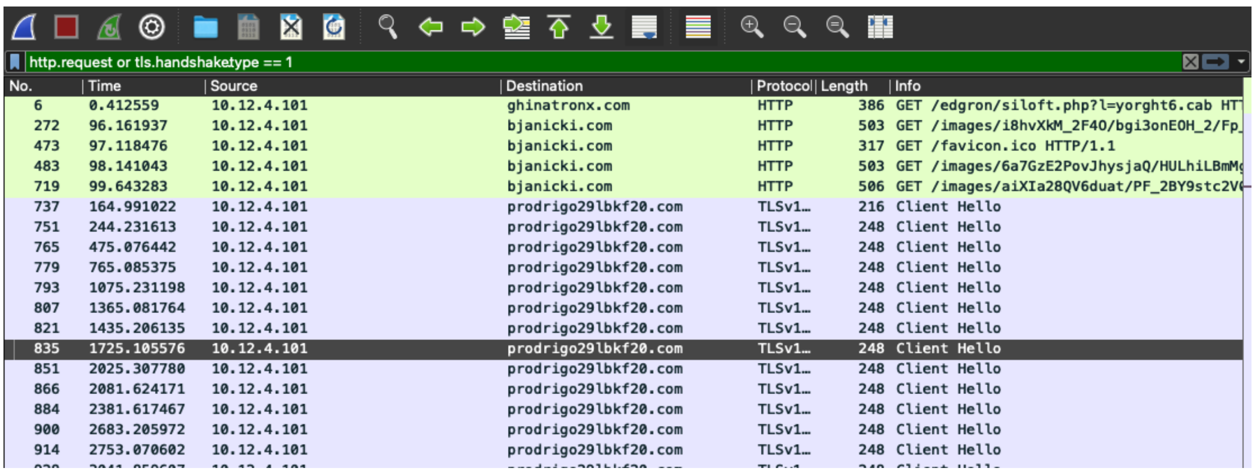
- Domain for initial GET requests: `foo.fulldin[.]at`
- Request for follow-up data: `hxxp://one.ahah100[.]at/jvassets/o1/s64.dat`
- Domain for GET and POST requests after Ursnif is persistent: `api.ahah100[.]at`

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.


http.request or tls.handshaketype == 1

No.	Time	Source	Destination	Protocol	Length	Info
6	0.412559	10.12.4.101	ghinatronx.com	HTTP	386	GET /edgron/siloft.php?l=yorght6.cab HT
272	96.161937	10.12.4.101	bjanicki.com	HTTP	503	GET /images/i8hvXkM_2F40/bgi3onEOH_2/Fp_
473	97.118476	10.12.4.101	bjanicki.com	HTTP	317	GET /favicon.ico HTTP/1.1
483	98.141043	10.12.4.101	bjanicki.com	HTTP	503	GET /images/6a7GzE2PovJhysjaQ/HULhiLBmM
719	99.643283	10.12.4.101	bjanicki.com	HTTP	506	GET /images/aiXIa28QV6duat/PF_2BY9stc2V
737	164.991022	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	216	Client Hello
751	244.231613	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
765	475.076442	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
779	765.085375	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
793	1075.231198	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
807	1365.081764	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
821	1435.206135	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
835	1725.105576	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
851	2025.307780	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
866	2081.624171	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
884	2381.617467	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
900	2683.205972	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
914	2753.070602	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello

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`.

```
GET /edgron/siloft.php?l=yorght6.cab HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: ghinatronx.com
Connection: Keep-Alive
```

```
MZ......................@..........................................  .!..L.!This program cannot be run in DOS mode.

$.......  lA_M
/.M
/.M
/.S_..W
/.S_..\
/.S_...
/.Du..J
/.M
...
/.Du..L
/.Du..L
/.Du..L
```

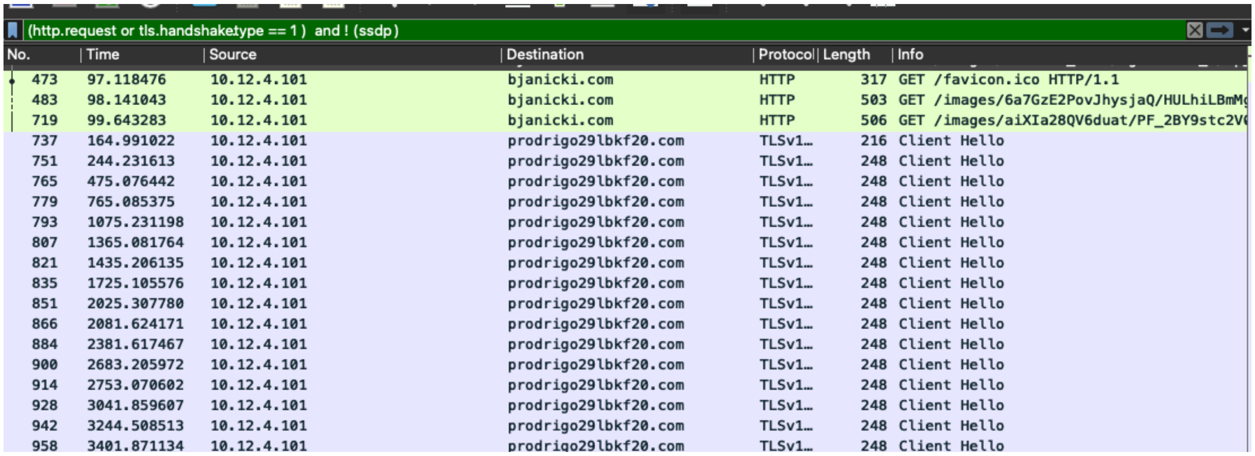
This TCP stream reveals a Windows executable or DLL file. As shown, MZ are two bytes of Windows EXE or DLL represented in ASCII format.

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.

```
GET /images/i8hvXkM_2F4O/bgi3onEOH_2/Fp_2FNWip7iwXT/I9ec6aw1_2BGhXbPixQHw/P7LK5Q_2Ft0TxcvC/wFLVNn_2By_2Fb2/WPHYci0rdY2dogSODh/YnkcDRKqk/
sQG3_2BH_2FAoIu48Zkg/4rkH7uEXf_2FnP0QxkH/sP_2BvAuw9PjX/ugTn.avi HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: bjanicki.com
DNT: 1
Connection: Keep-Alive
```

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`.



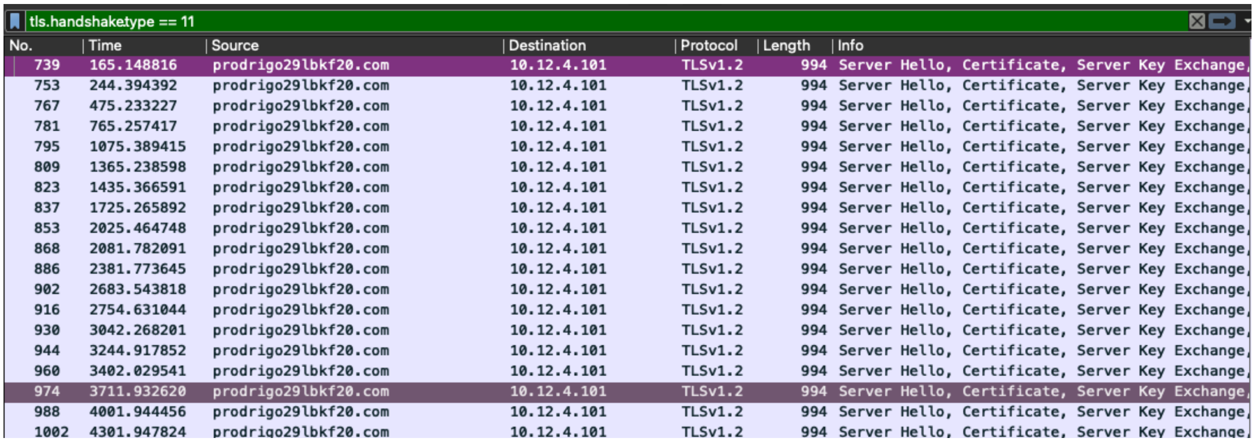
(http.request or tls.handshake.type == 1) and ! (ssdp)

No.	Time	Source	Destination	Protocol	Length	Info
473	97.118476	10.12.4.101	bjanicki.com	HTTP	317	GET /favicon.ico HTTP/1.1
483	98.141043	10.12.4.101	bjanicki.com	HTTP	503	GET /images/6a7GzE2PovJhysjaQ/HULhiLBmM
719	99.643283	10.12.4.101	bjanicki.com	HTTP	506	GET /images/aiXIa28QV6duat/PF_2BY9stc2V
737	164.991022	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	216	Client Hello
751	244.231613	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
765	475.076442	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
779	765.085375	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
793	1075.231198	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
807	1365.081764	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
821	1435.206135	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
835	1725.105576	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
851	2025.307780	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
866	2081.624171	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
884	2381.617467	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
900	2683.205972	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
914	2753.070602	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
928	3041.859607	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
942	3244.508513	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello
958	3401.871134	10.12.4.101	prodrigo29lbkf20.com	TLSv1…	248	Client Hello

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.

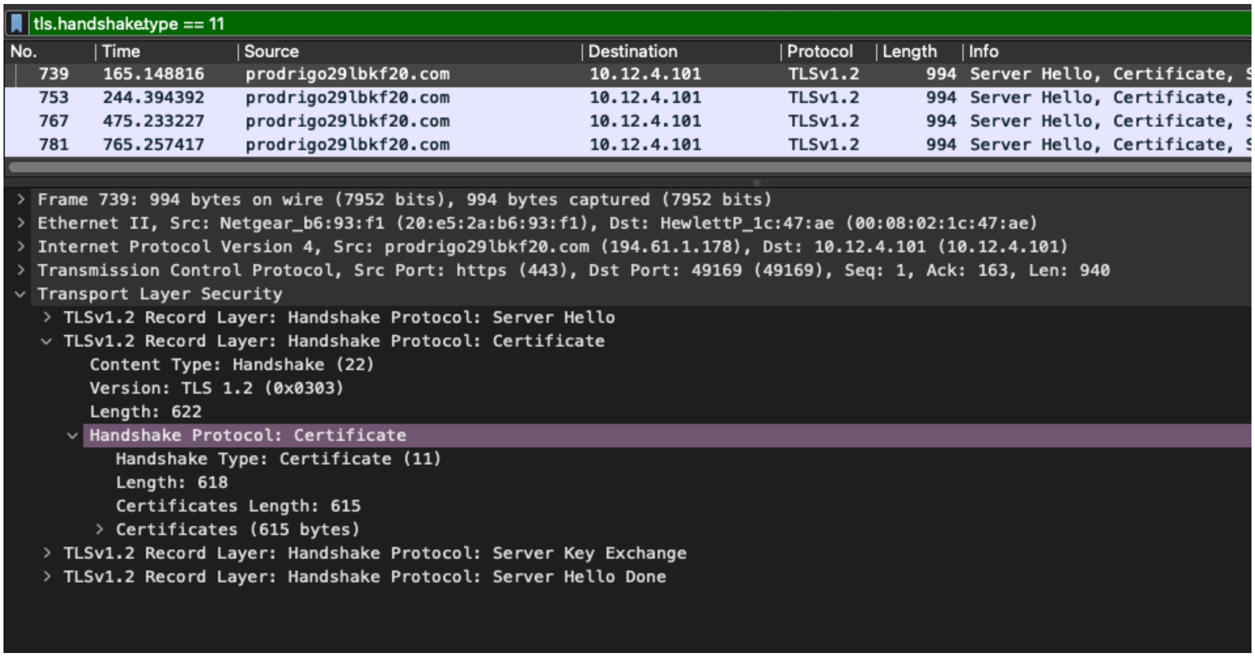
To get a closer look, filter on ***ssl.handshake.type == 11*** (or ***tls.handshake.type == 11*** in Wireshark 3.0 or newer).



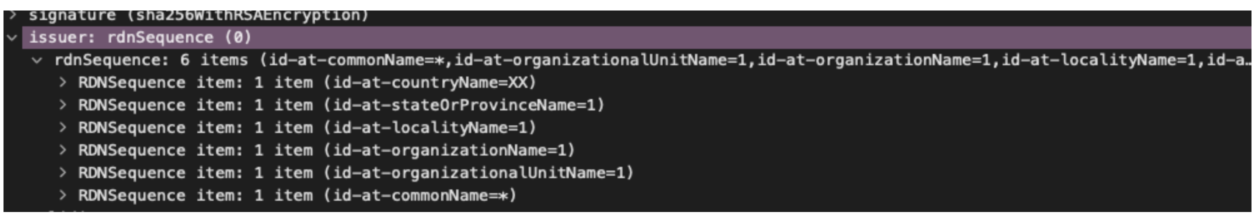
tls.handshake.type == 11

No.	Time	Source	Destination	Protocol	Length	Info
739	165.148816	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
753	244.394392	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
767	475.233227	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
781	765.257417	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
795	1075.389415	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
809	1365.238598	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
823	1435.366591	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
837	1725.265892	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
853	2025.464748	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
868	2081.782091	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
886	2381.773645	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
902	2683.543818	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
916	2754.631044	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
930	3042.268201	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
944	3244.917852	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
960	3402.029541	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
974	3711.932620	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
988	4001.944456	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,
1002	4301.947824	prodrigo29lbkf20.com	10.12.4.101	TLSv1.2	994	Server Hello, Certificate, Server Key Exchange,

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.



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:



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=*****

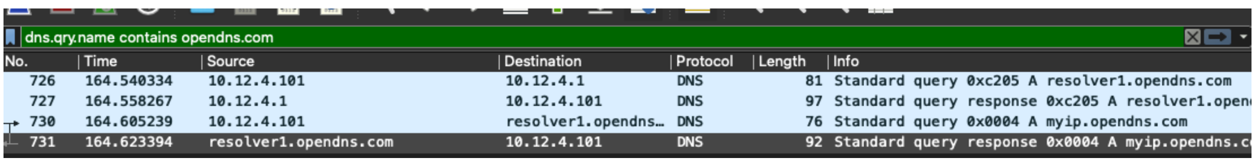
This issuer data is not valid, and these patterns are commonly seen in Ursnif infections.

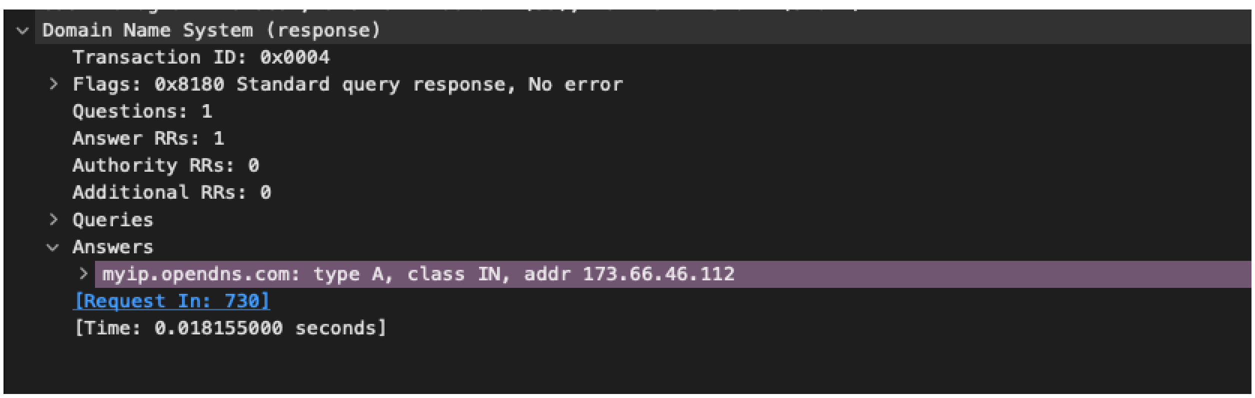
But what does legitimate certificate data look like? Below is a valid data from a certificate issued by DigiCert.

- 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.





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.

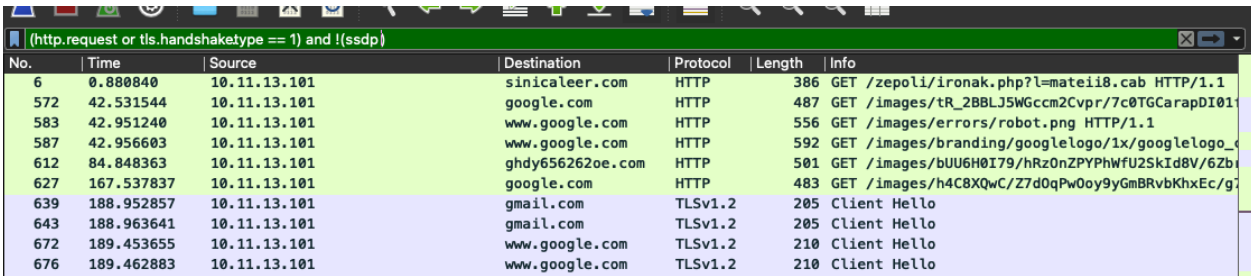
Example 3: Ursnif with Follow-up Malware

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.



(http.request or tls.handshake.type == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Length	Info
6	0.880840	10.11.13.101	sinicaleer.com	HTTP	386	GET /zepoli/ironak.php?l=mateii8.cab HTTP/1.1
572	42.531544	10.11.13.101	google.com	HTTP	487	GET /images/tR_2BBLJ5WGccm2Cvpr/7c0TGCarapDI01
583	42.951240	10.11.13.101	www.google.com	HTTP	556	GET /images/errors/robot.png HTTP/1.1
587	42.956603	10.11.13.101	www.google.com	HTTP	592	GET /images/branding/googlelogo/1x/googlelogo_
612	84.848363	10.11.13.101	ghdy656262oe.com	HTTP	501	GET /images/bUU6H0I79/hRzOnZPYPhWfU2SkId8V/6Zb
627	167.537837	10.11.13.101	google.com	HTTP	483	GET /images/h4C8XQwC/Z7d0qPwOoy9yGmBRvbKhxEc/g
639	188.952857	10.11.13.101	gmail.com	TLSv1.2	205	Client Hello
643	188.963641	10.11.13.101	gmail.com	TLSv1.2	205	Client Hello
672	189.453655	10.11.13.101	www.google.com	TLSv1.2	210	Client Hello
676	189.462883	10.11.13.101	www.google.com	TLSv1.2	210	Client Hello

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` :

```
GET /images/tR_2BBLJ5WGccm2Cvpr/7c0TGCarapDI01fwF90LUt/zBeIa00d1csb6/mKuc06Xs/0rQo1mpBWMzMhGYaWP5cGba/6cKKPu7M4k/upkGRgHtjyo_2FnIA/
YtgX2e1eVWz4/_2FIsETqwUK/i_2BnMfk/Yg85BsECNX/NAhr9.avi HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: google.com
DNT: 1
Connection: Keep-Alive
```

Note the HTTP traffic to `ghdy656262oe[.]com`.

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

```
HTTP/1.0 404 Not Found
Date: Wed, 13 Nov 2019 14:37:37 GMT
Server: Apache/2.4.6 (CentOS) PHP/5.4.16
X-Powered-By: PHP/5.4.16
Set-Cookie: PHPSESSID=l7cer7uotsdfksqkr27mdvu2k3; path=/; domain=.ghdy656262oe.com
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: lang=en; expires=Fri, 13-Dec-2019 14:37:37 GMT; path=/; domain=.ghdy656262oe.com
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

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

```
HTTP/1.1 200 OK
Date: Wed, 13 Nov 2019 14:41:50 GMT
Server: Apache/2.4.6 (CentOS) PHP/5.4.16
X-Powered-By: PHP/5.4.16
Set-Cookie: PHPSESSID=f0hg6i0oqe50v6e0q31e7f9bu4; path=/; domain=.ghdy656262oe.com
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

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***

((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

No.	Time	Source	Destination	Protocol	Length	Info
612	84.848363	10.11.13.101	ghdy656262oe.com	HTTP	501	GET /images/bUU6H0I79/hRzOnZPYPhWfU2SkId8V/6Zbr_
614	85.149236	ghdy656262oe.com	10.11.13.101	HTTP	578	HTTP/1.0 404 Not Found
618	106.409326	10.11.13.101	10.11.13.1	DNS	74	Standard query 0x2f94 A tnzf3380au.top
619	106.604433	10.11.13.1	10.11.13.101	DNS	144	Standard query response 0x2f94 No such name A tn
620	136.907407	10.11.13.101	10.11.13.1	DNS	73	Standard query 0x7871 A xijamaalj.com
621	137.087900	10.11.13.1	10.11.13.101	DNS	146	Standard query response 0x7871 No such name A xi
723	211.209492	10.11.13.101	ghdy656262oe.com	HTTP	507	GET /images/CodzxTtiCg4nrPiJ51ry/uxx1fhPtfuqXbTk
725	211.515593	ghdy656262oe.com	10.11.13.101	HTTP	484	HTTP/1.0 404 Not Found
783	338.290772	10.11.13.101	ghdy656262oe.com	HTTP	516	GET /images/qMY4GHlb/4hvJbIBd5m3ZUaqnXz4nNWS/kWc
1065	339.552040	ghdy656262oe.com	10.11.13.101	HTTP	1245	HTTP/1.1 200 OK (text/html)
1067	339.574903	10.11.13.101	ghdy656262oe.com	HTTP	321	GET /favicon.ico HTTP/1.1
1074	339.884669	ghdy656262oe.com	10.11.13.101	HTTP	225	HTTP/1.1 200 OK (image/vnd.microsoft.icon)
1079	341.870548	10.11.13.101	ghdy656262oe.com	HTTP	504	GET /images/i3i0OQaVO382xXNupBTPME/vSCSw5eRjs2Wh
1440	343.573363	ghdy656262oe.com	10.11.13.101	HTTP	1055	HTTP/1.1 200 OK (text/html)
1442	344.635499	10.11.13.101	ghdy656262oe.com	HTTP	519	GET /images/fbF27qOoXF/9Nlp42ISAcE_2BLWS/nbzeBPN
1448	344.792610	10.11.13.101	ghdy656262oe.com	HTTP	519	GET /images/fbF27qOoXF/9Nlp42ISAcE_2BLWS/nbzeBPN
1452	345.036504	ghdy656262oe.com	10.11.13.101	HTTP	147	HTTP/1.1 200 OK (text/html)

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**:

1468	410.491568	10.11.13.101	google.com	TLSv1.2	206	Client Hello
1489	410.845632	10.11.13.101	gmail.com	TLSv1.2	205	Client Hello
1510	458.093907	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	215	Client Hello
1525	459.732083	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1544	461.494749	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1563	463.123988	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1597	711.121036	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1614	712.845799	10.11.13.101	carresqautomotive…	HTTP	236	GET /jjwekr.rar HTTP/1.1

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`.

783	338.290772	10.11.13.101	ghdy656262oe.com	HTTP	516	GET /images/qMY4GHlb/4hvJbIBd5m3
1067	339.574903	10.11.13.101	ghdy656262oe.com	HTTP	321	GET /favicon.ico HTTP/1.1
1079	341.870548	10.11.13.101	ghdy656262oe.com	HTTP	504	GET /images/i3i0OQaVO382xXNupBTP
1442	344.635499	10.11.13.101	ghdy656262oe.com	HTTP	519	GET /images/fbF27qOoXF/9Nlp42ISA
1448	344.792610	10.11.13.101	ghdy656262oe.com	HTTP	519	GET /images/fbF27qOoXF/9Nlp42ISA
1468	410.491568	10.11.13.101	google.com	TLSv1.2	206	Client Hello
1489	410.845632	10.11.13.101	gmail.com	TLSv1.2	205	Client Hello
1510	458.093907	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	215	Client Hello
1525	459.732083	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1544	461.494749	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1563	463.123988	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1597	711.121036	10.11.13.101	vnt69tnjacynthe.c…	TLSv1.2	247	Client Hello
1614	712.845799	10.11.13.101	carresqautomotive…	HTTP	236	GET /jjwekr.rar HTTP/1.1

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`.

```
GET /jjwekr.rar HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Win64; x64)
Host: carresqautomotive.com
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Wed, 13 Nov 2019 14:48:05 GMT
Server: Apache
Last-Modified: Wed, 13 Nov 2019 12:17:18 GMT
Accept-Ranges: bytes
Content-Length: 196679
Content-Type: application/x-rar-compressed
```

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


```
 .....y.i.6.....s..b...e.m...{.........bY..a&+]K....Q.1E:...D....Ph.w./......|
m.8..      .j.K.kp
y..69.Pk.4......'.O...X2..........xo....Jt&5......9.,Q.......gS..1T.:." |.j..3I.BA......@L..z...wE^..")....B.O....{.a_0.Eq..K ...
\4,.yY|.Bn4R...J.r..}....B.....T..Q.0..LW....1.....C~.......{.O.I....0 ......|..........>sl.k..c....yd....G...
....Y=.S.....!.qcVk.GG=..J/H.]RH._.U.=R=.e.wpa7.<......R.. ............4.........vW.
q..>.K....8....9u...........ua.).._.f5[i....|....B...x.?.;.   W.\.^O....W~..W.Yg'..T.R./....X....(..3..E..-..X.....c..XE.L+|%.w...
..I.=.... ..[...~.M..|.        .~.=..3m...@.v..B..Y.g..&
.h.
...*;....vw.../........e*i....4 +>t...w...C.....7..+^........)ao%...M..G.d
9NJ.....4$......KHS.....'v....*....W{...@...*%..XQ.a`............a.....\...sL...AE...=.jR........|>T.%~.....
...%......... H#..#:.HX.F."{...T..[.....I^..nx.*."....us..5.j..%..}......$.....x....S.i.....9..j..[{.'.J.~.\...\s..e?.5..Z;...L~.b}"....
{|d.+...Sd1..=h.l....uO....=...D[q2.V..Hw.A.Rv.'..p......LC.....?....
X*0.[9 3s..?.....}h.<kt.h...5..(U....._O.3....H..`.i;..7.\...E..m..>....wAA6.:...~.U.|.^.w.t.6-..o.T.\iL.....h....;.;.........I..Ty._J.fC.
8.......J'.o....%`..KNg..yS|9!..
Q.Q...(}}...0....S.z..O.~.I..C..-8.B..nf..m.M;...0.lt0.2..s.+:K......m...N..      .r..~I7..v.n.....8.7 ..LH.:l.....R._;..o....|..A.
5.A..C..ur3....ud.............[.r~]_7a.
....       .3W.....0.hS..T....e...f...c.>:yX..8...t|2_...%K..ux..w.%......S...J.w#.I........I..0....6Wp...,..X;O.k.c............._...]. ...
5,..My...../...|....B2.
.<...g+4t.S.3...~3..6......\....f.v...Y...8..,./2.s....te....W..g.u.*.)..(.....s.i'%
o.....N..F@.F.Z.N...X.....PZ...*....p.*.f..l............t#~...q..".\_.7..... .9x{Mn....,
?.q..R7....yO..m.........W`..a...!....3......#.GI8...9U..6.....\...H.d....<..........^.x..Fs..!P+HP....          .f.
.)...b&u..uY...%gzj....H(.4-Y....M{..~..*...:%.0.'..I.;F..uh.(..1D).p...z~..f*. ..|.....pO....5..*WO.
```


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, IcedID, Nymain, Pushdo, and Trickbot. 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.

(http.request or tls.handshaketype == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Length	Info
47	14.353170	10.11.12.101	a1961.g2.akamai.n…	HTTP	151	GET /ncsi.txt HTTP/1.1
71	40.837541	10.11.12.101	oklogallem.com	HTTP	386	GET /zepoli/ironak.php?l=luntsu1.cab HTTP/1.1
323	66.709338	10.11.12.101	google.com	HTTP	787	GET /images/SPdsgBJ5WiV_2BAGp5Z/kN8cgY1azSH7U4F
337	66.882285	10.11.12.101	www.google.com	HTTP	856	GET /images/errors/robot.png HTTP/1.1
339	66.882791	10.11.12.101	www.google.com	HTTP	892	GET /images/branding/googlelogo/1x/googlelogo_
443	96.680813	10.11.12.101	cs9.wpc.v0cdn.net	TLSv1.2	220	Client Hello
445	96.680905	10.11.12.101	cs9.wpc.v0cdn.net	TLSv1.2	218	Client Hello
447	96.680972	10.11.12.101	cs9.wpc.v0cdn.net	TLSv1.2	220	Client Hello
449	96.681151	10.11.12.101	cs9.wpc.v0cdn.net	TLSv1.2	218	Client Hello
553	109.150569	10.11.12.101	kh2714ldb.com	HTTP	516	GET /images/58HuD8VcxhOH06K/eUWS28C7Jfyw4oHXzL/
557	109.690919	10.11.12.101	kh2714ldb.com	HTTP	318	GET /favicon.ico HTTP/1.1
634	283.145088	10.11.12.101	google.com	HTTP	786	GET /images/z5FTh4xviE9DaSwTyQW/XXFK421uhrO_2By
651	304.375426	10.11.12.101	gmail.com	TLSv1.2	205	Client Hello
655	304.383316	10.11.12.101	gmail.com	TLSv1.2	205	Client Hello
685	304.574452	10.11.12.101	www.google.com	TLSv1.2	210	Client Hello
689	304.580962	10.11.12.101	www.google.com	TLSv1.2	210	Client Hello

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`.

71	40.837541	10.11.12.101	oklogallem.com	HTTP	386	GET /zepoli/ironak.php?l=luntsu1.cab HTTP/1.1

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

553	109.150569	10.11.12.101	kh2714ldb.com	HTTP	516	GET /images/58HuD8VcxhOH06K/eUWS28C7Jfyw4oHXzL/
557	109.690919	10.11.12.101	kh2714ldb.com	HTTP	318	GET /favicon.ico HTTP/1.1

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

1299	406.395002	10.11.12.101	s9971kbjjessie.com	TLSv1.2	214	Client Hello
1336	694.881808	10.11.12.101	s9971kbjjessie.com	TLSv1.2	246	Client Hello
1468	994.883997	10.11.12.101	s9971kbjjessie.com	TLSv1.2	246	Client Hello
1483	996.202643	10.11.12.101	s9971kbjjessie.com	TLSv1.2	246	Client Hello

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.

1500	998.793496	10.11.12.101	startuptshirt.my	HTTP	261	GET /wp-content/uploads/2019/11/jjasndeqw.rar
1755	1003.587254	10.11.12.101	94.140.114.6	TLSv1.2	187	Client Hello
1779	1005.913527	10.11.12.101	94.140.114.6	TLSv1.2	219	Client Hello
2287	1009.772737	10.11.12.101	94.140.114.6	TLSv1.2	219	Client Hello
2312	1015.802697	10.11.12.101	94.140.114.6	TLSv1.2	219	Client Hello
2359	1267.232396	10.11.12.101	5.61.34.51	TLSv1	187	Client Hello
2388	1283.283478	10.11.12.101	5.61.34.51	TLSv1	181	Client Hello
2417	1287.505576	10.11.12.101	5.61.34.51	TLSv1	181	Client Hello
2436	1290.849854	10.11.12.101	5.61.34.51	TLSv1	181	Client Hello

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***

(ip.addr eq 94.140.114.6 or ip.addr eq 5.61.34.51) and tls.handshaketype eq 11

No.	Time	Source	Destination	Protocol	Length	Info
1757	1003.747666	94.140.114.6	10.11.12.101	TLSv1.2	1155	Server Hello, Certificate, Server Hello Done
1781	1006.079577	94.140.114.6	10.11.12.101	TLSv1.2	1155	Server Hello, Certificate, Server Hello Done
2289	1009.937305	94.140.114.6	10.11.12.101	TLSv1.2	1155	Server Hello, Certificate, Server Hello Done
2314	1015.967392	94.140.114.6	10.11.12.101	TLSv1.2	1155	Server Hello, Certificate, Server Hello Done
2361	1267.423167	5.61.34.51	10.11.12.101	TLSv1	1189	Server Hello, Certificate, Server Hello Done
2390	1283.590699	5.61.34.51	10.11.12.101	TLSv1	1189	Server Hello, Certificate, Server Hello Done
2419	1287.702139	5.61.34.51	10.11.12.101	TLSv1	1189	Server Hello, Certificate, Server Hello Done
2438	1291.048342	5.61.34.51	10.11.12.101	TLSv1	1189	Server Hello, Certificate, Server Hello Done

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

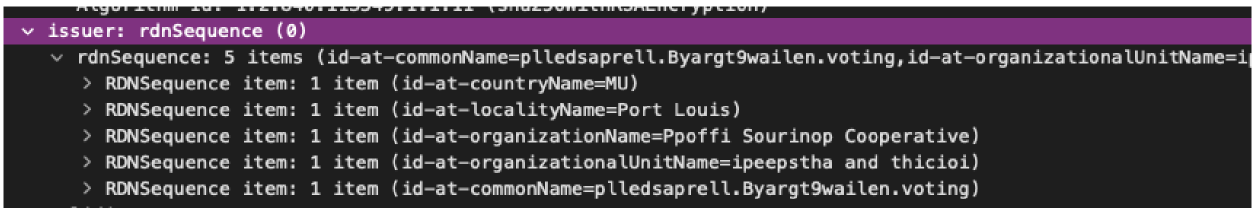
```
v issuer: rdnSequence (0)
  v rdnSequence: 5 items (id-at-commonName=ndltman-dsamutb.spiegel,id-at-organizationalUnitName=Olfo Du
    > RDNSequence item: 1 item (id-at-countryName=NP)
    > RDNSequence item: 1 item (id-at-localityName=Kathmandu)
    > RDNSequence item: 1 item (id-at-organizationName=Buvecoww Fftaites O.V.E.E.)
    > RDNSequence item: 1 item (id-at-organizationalUnitName=Olfo Dusar Latha)
    > RDNSequence item: 1 item (id-at-commonName=ndltman-dsamutb.spiegel)
```

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`:



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**.

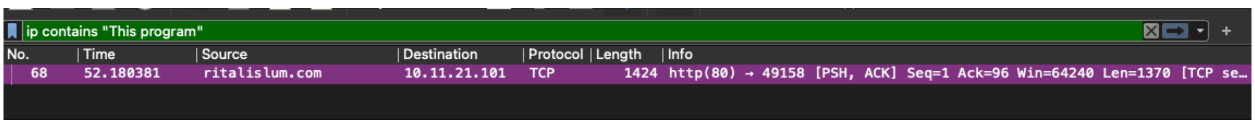
Example 5: Evaluation

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

⇒ 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*

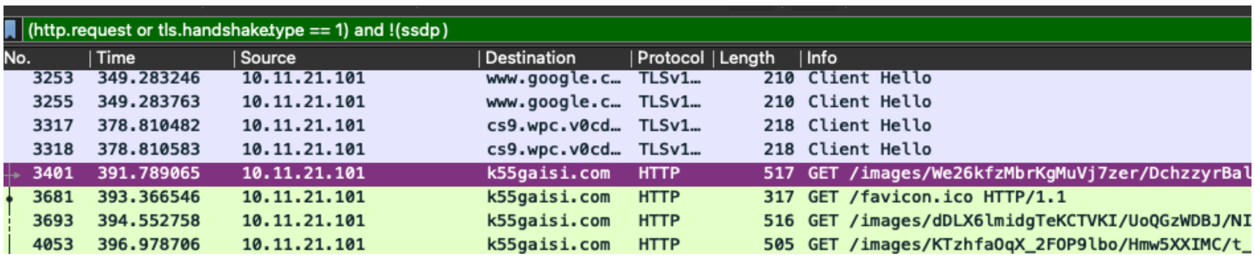
After the initial Ursnif binary was sent, the infected Windows host contacted different domains for the HTTP GET requests. Which domain was the traffic successful and allowed the infection to proceed?

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.



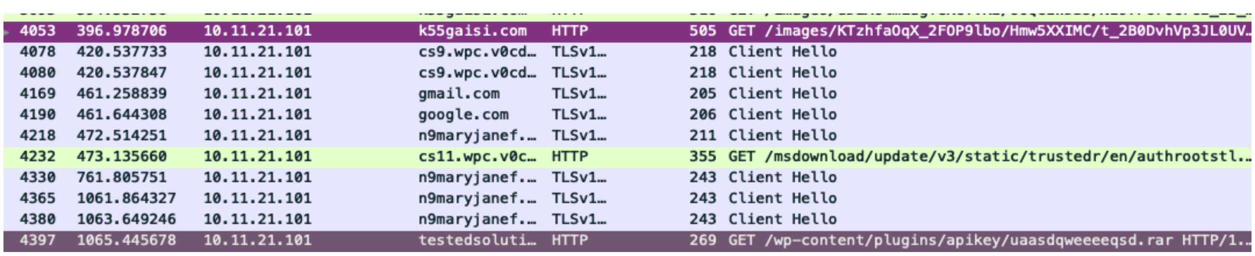
(http.request or tls.handshake.type == 1) and !(ssdp)

No.	Time	Source	Destination	Protocol	Length	Info
3253	349.283246	10.11.21.101	www.google.c…	TLSv1…	210	Client Hello
3255	349.283763	10.11.21.101	www.google.c…	TLSv1…	210	Client Hello
3317	378.810482	10.11.21.101	cs9.wpc.v0cd…	TLSv1…	218	Client Hello
3318	378.810583	10.11.21.101	cs9.wpc.v0cd…	TLSv1…	218	Client Hello
3401	391.789065	10.11.21.101	k55gaisi.com	HTTP	517	GET /images/We26kfzMbrKgMuVj7zer/DchzzyrBal
3681	393.366546	10.11.21.101	k55gaisi.com	HTTP	317	GET /favicon.ico HTTP/1.1
3693	394.552758	10.11.21.101	k55gaisi.com	HTTP	516	GET /images/dDLX6lmidgTeKCTVKI/UoQGzWDBJ/NI
4053	396.978706	10.11.21.101	k55gaisi.com	HTTP	505	GET /images/KTzhfa0qX_2FOP9lbo/Hmw5XXIMC/t_

⇒ `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?

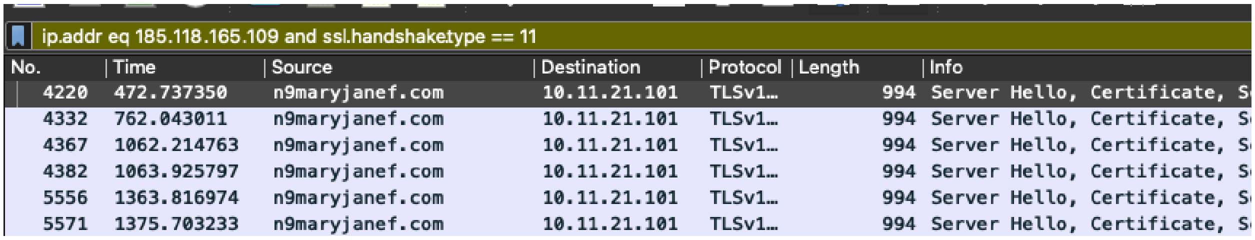
When Ursnif is persistent, we no longer see Ursnif-style HTTP requests starting with `GET /images/`. Instead, we find Ursnif-related HTTPS traffic. Shortly after the final Ursnif-style HTTP GET request, HTTPS traffic to `n9maryjanef[.]com` begins as highlighted in the Figure.



4053	396.978706	10.11.21.101	k55gaisi.com	HTTP	505	GET /images/KTzhfa0qX_2FOP9lbo/Hmw5XXIMC/t_2B0DvhVp3JL0UV…
4078	420.537733	10.11.21.101	cs9.wpc.v0cd…	TLSv1…	218	Client Hello
4080	420.537847	10.11.21.101	cs9.wpc.v0cd…	TLSv1…	218	Client Hello
4169	461.258839	10.11.21.101	gmail.com	TLSv1…	205	Client Hello
4190	461.644308	10.11.21.101	google.com	TLSv1…	206	Client Hello
4218	472.514251	10.11.21.101	n9maryjanef.…	TLSv1…	211	Client Hello
4232	473.135660	10.11.21.101	cs11.wpc.v0c…	HTTP	355	GET /msdownload/update/v3/static/trustedr/en/authrootstl…
4330	761.805751	10.11.21.101	n9maryjanef.…	TLSv1…	243	Client Hello
4365	1061.864327	10.11.21.101	n9maryjanef.…	TLSv1…	243	Client Hello
4380	1063.649246	10.11.21.101	n9maryjanef.…	TLSv1…	243	Client Hello
4397	1065.445678	10.11.21.101	testedsoluti…	HTTP	269	GET /wp-content/plugins/apikey/uaasdqweeeeqsd.rar HTTP/1…

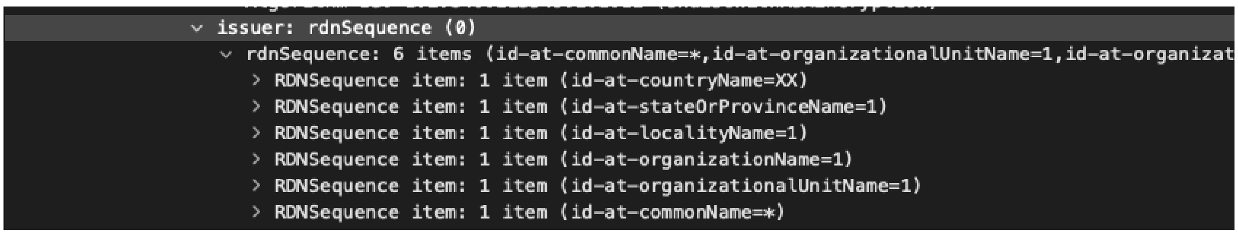
⇒ This is Ursnif traffic. The domain is `n9maryjanef[.]com`.

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



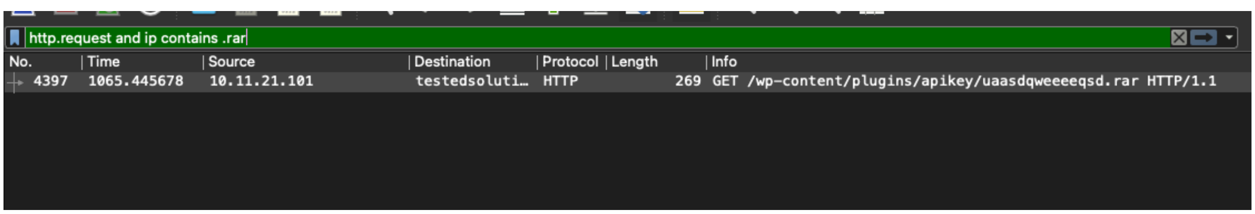
No.	Time	Source	Destination	Protocol	Length	Info
4220	472.737350	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S
4332	762.043011	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S
4367	1062.214763	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S
4382	1063.925797	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S
5556	1363.816974	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S
5571	1375.703233	n9maryjanef.com	10.11.21.101	TLSv1…	994	Server Hello, Certificate, S

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



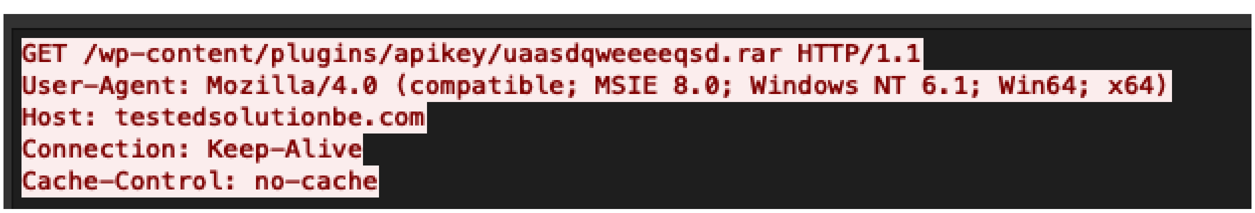
What URL ending in .rar was **used to send follow-up malware** to the infected Windows host?

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***



No.	Time	Source	Destination	Protocol	Length	Info
4397	1065.445678	10.11.21.101	testedsoluti…	HTTP	269	GET /wp-content/plugins/apikey/uaasdqweeeeqsd.rar HTTP/1.1

The results are as follow:



```
GET /wp-content/plugins/apikey/uaasdqweeeeqsd.rar HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Win64; x64)
Host: testedsolutionbe.com
Connection: Keep-Alive
Cache-Control: no-cache
```

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 21 Nov 2019 15:51:44 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Cache-Control: max-age=3600
Expires: Thu, 21 Nov 2019 16:51:44 GMT
Location: https://testedsolutionbe.com/wp-content/plugins/apikey/uaasdqweeeeqsd.rar
Server: cloudflare
CF-RAY: 5393d3a4fbbd2645-DFW
```

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:

(http.request or tls.handshake.type == 1) and ! (ssdp)

No.	Time	Source	Destination	Protocol	Length	Info
4330	761.805751	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
4365	1061.864327	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
4380	1063.649246	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
4397	1065.445678	10.11.21.101	testedsoluti…	HTTP	269	GET /wp-content/plugins/apikey/uaasdqweeeeqsd.rar HTTP/1…
4404	1065.651923	10.11.21.101	testedsoluti…	TLSv1…	216	Client Hello
4735	1069.239420	10.11.21.101	185.99.133.38	TLSv1…	187	Client Hello
4761	1072.612262	10.11.21.101	185.99.133.38	TLSv1…	219	Client Hello
5418	1078.294136	10.11.21.101	185.99.133.38	TLSv1…	219	Client Hello
5441	1084.914602	10.11.21.101	185.99.133.38	TLSv1…	219	Client Hello
5482	1291.788515	10.11.21.101	5.61.34.51	TLSv1	187	Client Hello
5511	1312.129488	10.11.21.101	5.61.34.51	TLSv1	181	Client Hello
5532	1316.100405	10.11.21.101	5.61.34.51	TLSv1	181	Client Hello
5554	1363.583851	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5569	1375.452868	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5601	1662.820744	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5616	1665.738158	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5659	1965.680582	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5682	2035.393452	10.11.21.101	gmail.com	TLSv1…	237	Client Hello
5699	2038.013034	10.11.21.101	google.com	TLSv1…	238	Client Hello
5715	2038.453729	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5734	2044.608283	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5771	2261.890170	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5786	2265.786248	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5808	2279.913605	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello
5841	2566.211297	10.11.21.101	n9maryjanef…	TLSv1…	243	Client Hello

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

[3] Using Wireshark – Display Filter Expressions

[4] Using Wireshark: Identifying Hosts and Users

[5] Using Wireshark: Exporting Objects from a Pcap

[6] Wireshark Tutorial: Examining Trickbot Infections