TinyPOS and ProLocker: An Odd Relationship

norfolkinfosec.com/tinypos-and-prolocker-an-odd-relationship/

norfolk November 2, 2020

Recently, I came across a VISA bulletin regarding point-of-sale malware being used against merchant targets. In Incident #1 in this VISA report, VISA described a deployment technique for TinyPOS that seemed oddly similar to the ProLocker ransomware installation workflow described by Group-IB, although I initially dismissed this as a coincidence.

After spending time mapping out code-level relationships and VirusTotal submitter relationships (initially with the intent of identifying an entry vector), there is evidence to suggest that this is *not* pure chance. In short, one of the following is likely true:

1. ProLocker and TinyPOS are written by the same author, who *also* provides a deployment mechanism; or,

2. ProLocker and TinyPOS are written, deployed, and used by the same threat actor

3. The ProLocker adversary obtained or modified the TinyPOS source code and also operates in the carding space

Of these, the second seems the most likely. In addition to distinct code-level relationships shared across several tools from both threat actors (and no apparent other threat actors) *and* the very similar delivery mechanisms, both ProLocker attacks and TinyPOS attacks appear to be low-volume enough that it is plausible a single small to medium-sized group is operating them, rather than two distinct entities. This would parallel assessments that other threat groups who traditionally operated in the carding and banking spaces have also switched to ransomware attacks, including FIN6 and TA505.

This remainder of this post primarily walks through the analytic workflow that led to these assessments (as opposed to a traditional intelligence-style condensed publication of the key facts) so that others may properly evaluate the methodology and findings.

I.Identifying More Files

TinyPOS Installation Workflow

Incident #1 in the VISA report contains two sets of indicators of compromise (IOC) including filenames and hashes, which describe the following installation workflow:

1. A .bat file executes a PowerShell script
2. This PowerShell script reads data “hidden” inside an image file
3. This data is loaded and run in memory as shellcode

4. The shellcode decodes itself into the TinyPOS malware

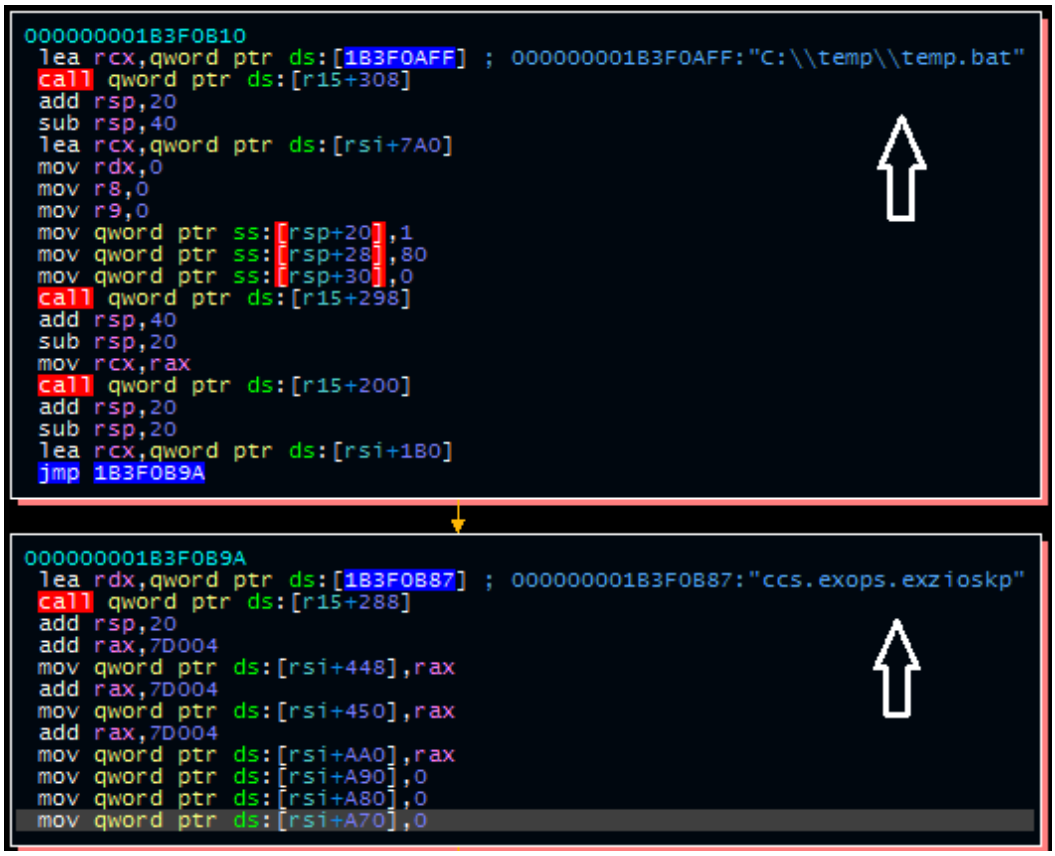
There are a few items worth noting here. First, VISA provided two examples of such image files from two observed installation workflows, one of which is located at “c:\temp\” and the other at the “c:\journal\” directory. The second item worth noting is that while the shellcode is simply “appended” to the image file and not obscured through a more complex steganography method. These characteristics will be important for the ProLocker comparisons.

The files for the hashes provided are not on VirusTotal and VISA offers limited additional details regarding these files; however, a report from Carbon Black describes the exact same files and provides more information, including code snippets for a decoded version of the PowerShell loader and the shellcode.

Obtaining a Copy of The Shellcode

At this point, there is actually enough information to identify a copy of the shellcode on VirusTotal. Figure 4 in Carbon Black’s report highlights the start of the shellcode – by taking some of these bytes and running a VirusTotal content search (e.g. content:”{48 31 Db 48 C7 C0 A3 A1 A8}”), a single result for a file named “t.bin” appears:

MD5: f1efe5959ac5f730e08fb629143a78f9
SHA1: 6544e7506163782ccb2e06348d3c9467d0513be9
SHA256: 0be35b73262e67569e02950f5de9b94b7e0915dcd8ef4d8de66a6db600e41a18
Debugged, this file contains a decoding routine at the start that unpacks the rest of the shellcode, resulting in the TinyPOS sample identical to the one described by VISA and Carbon Black:



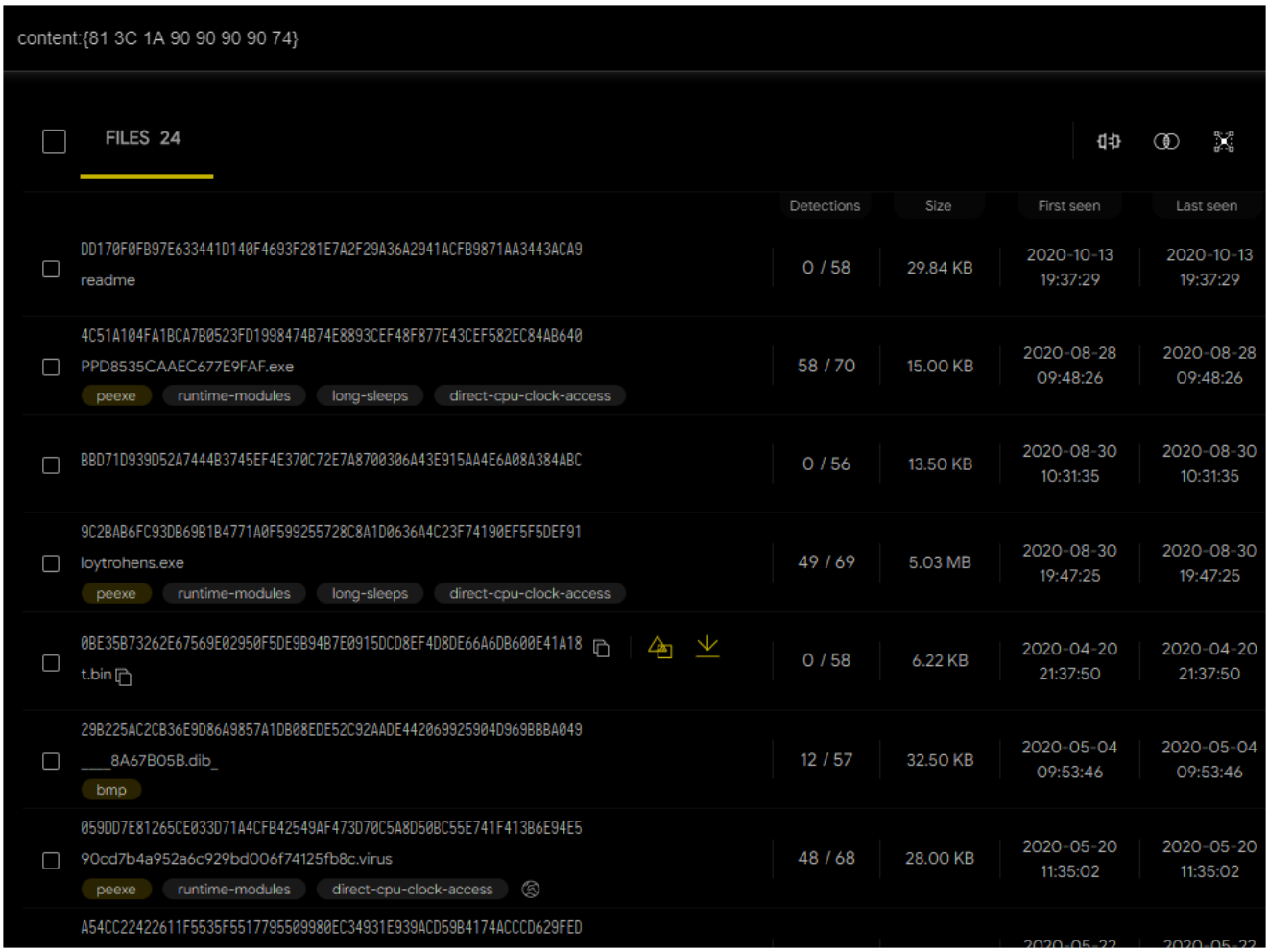
Decoded/Unpacked shellcode matching Carbon Black report

With a confirmed copy of this shellcode in hand, there are two additional pivoting vectors. First, there is a pivoting opportunity using the *decoded* segments of TinyPOS. Taking a set of opcode, a content search (content:"{41 ff 57 38 48 83 C4 20 49 89 87 00 02}") yielded two additional hashes:

MD5: a88107d4723ee6e6b33eca655c68a562 (sql.exe)
MD5: 57acabff815119ac5c391adbf8133c4a (sqlsrv.exe)

Each of these is an additional TinyPOS sample, but in executable form. Further pivoting from one of these submitters led to a TinyLoader sample uploaded alongside the TinyPOS sample communicating with known infrastructure from this threat actor, but this became a dead end for further meaningful research.

On the other hand, pivoting using a different part of the initial decoding function yielded far more interesting results: 24 hashes, all of which (with the exception of a single unclassified file) are ProLocker ransomware components or appear to be TinyPOS-related files.



The next section examines these files.

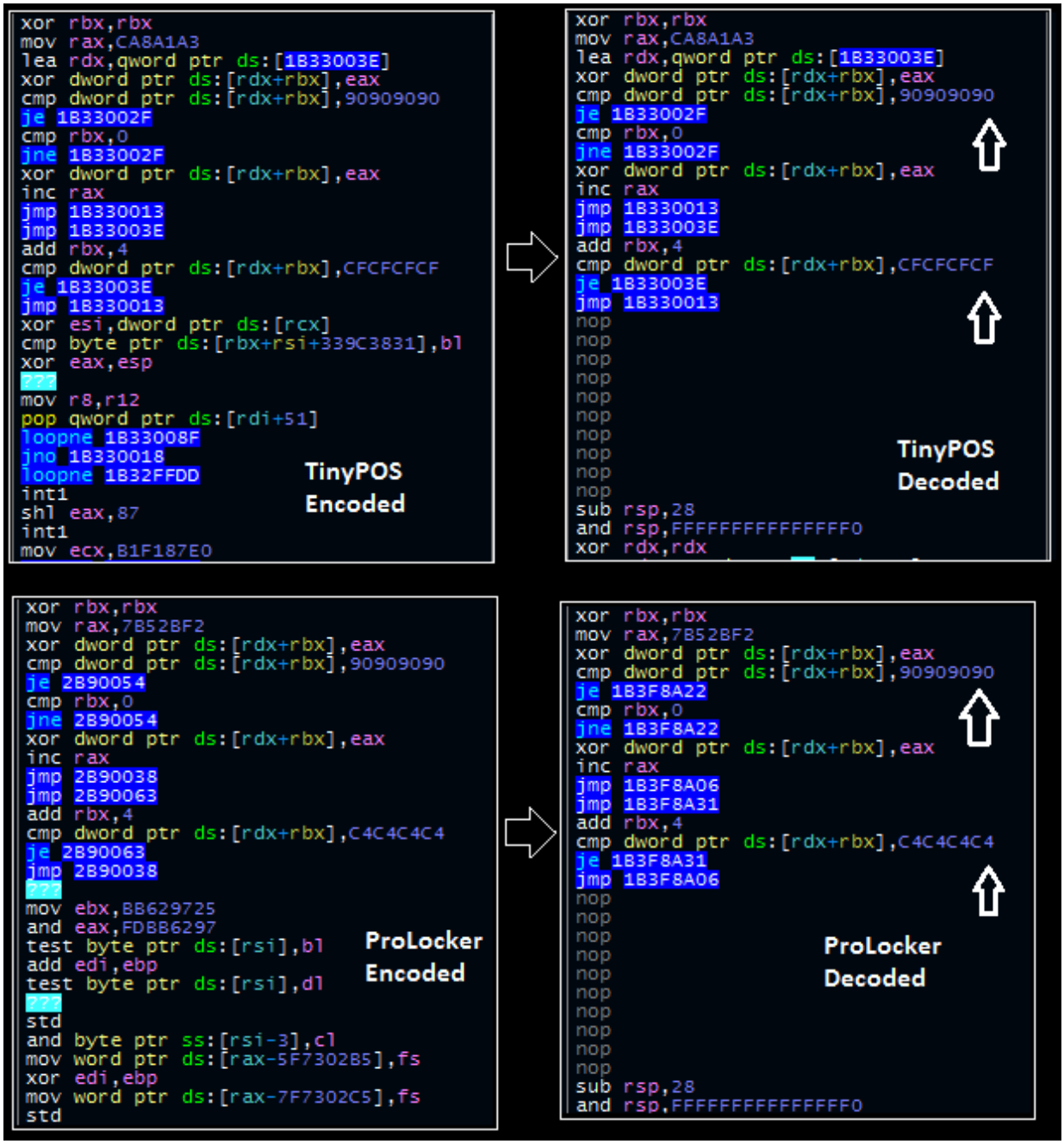
II.ProLocker and TinyPOS Relationship

The following hashes were identified and classified by pivoting with a VirusTotal content search using opcode from the decoding algorithm:

Hash	Filename	Description
F1EFE5959AC5F730E08FB629143A78F9	t.bin	TinyPOS (Previously identified)
5FC14B914B9A7DC2D546BC33E4B80584	ccv_server.txt	TinyPOS payload
417F35F30BCB474340CDB3F50491C1B0	readme_shellcode.exe	TinyPOS behavior
74D5D09F513FD5EBEE1EFF50495AC2D8	SQLSRV.EXE	TinyPOS behavior
E9C27A9F2A221527E03C36917DC36CB9	SQLSRV.EXE	TinyPOS behavior

Hash	Filename	Description
2A673709121D05BC57863002F8C62C51	PPD8535CAAEC677E9FAF.exe	ProLocker
16A29314E8563135B18668036A6F63C8	–	ProLocker
F3634A3B184B68A10C7A4849D378171A	7ZSfxMod	ProLocker
404EF54232F1817BA4258392815E1D22	NAS.exe	ProLocker
C182610DD437F90D0CC6CB0AC19CFDB7	loytrohens.exe	ProLocker
FE659D877AED2178EF084E3BF1E40254	MCC1D3C303AEA0018852.exe	ProLocker
02C01B59D0621815FC6A367FB1C7474E	LOCK.exe	ProLocker
AE3AAB90F69A05B131BD76ABE8A5A988	MCC1D3C303AEA0018852.exe	ProLocker
3355ACE345E98406BDB331CCAD568386	NAS_0.exe	ProLocker
90CD7B4A952A6C929BD006F74125FB8C	–	ProLocker
B0EEEC6DCA9F208C3E2B43EBF26D80BA	lock.exe	ProLocker
7AD4AFD690A1C69356BB3D0C8AD0947B	WRFF965C1.jpeg	ProLocker code
C579341F86F7E962719C7113943BB6E4	Winmgr.bmp	ProLocker code
B77EAE27DB59E660F972FAB37708807F	___8A67B05B.dib_	ProLocker code
34525178FB98B59E9BD98DF1ABE58C28	MCC1-D3C3-03AE-A001-8852.db	ProLocker code
BC469BF7946B9153D6270551F554B839	2B9E5820.db	ProLocker code
34F8DC1C8A0B49627B118C21E7C3B047	readme	ProLocker code
EA6E664A4EADE0428E6CD10028C9F3A7	readme	File containing code with TinyPOS behavior
D28AD0CC48005A09A04BA1D95275EE9A	–	Unknown

These file paint an important picture: the *only* files on VirusTotal with these opcode patterns belong to one of these two families. There are slight differences, including a different decoding key across files and an added instruction, but the core decoding mechanism and the "noop" buffer are identical.



These files also share another interesting property: when using a list to check processes, both malware families use a list consisting of partial process names that are six characters long. For example, the t.bin TinyPOS sample has a target list for "ccs.ex", "ops.ex", and "zioskp". The first two may be related to Oracle point-of-sale components, and the latter a Ziosk product. ProLocker uses the exact same type of process list, albeit with a different purpose (closing processes that might interfere with document and database information).

While compelling, these characteristics alone would be insufficient information to strongly assess a common operator (rather than simply a common author). However, there are additional factors to consider. The staging locations appear to be relatively consistent across these files. The TinyPOS shellcode files appear to be staged in subdirectories of the C:\ root drive, such as “c:\journal\”, “c:\temp\”, and (per a reddit post) “c:\Windows\”. For the ProLocker files, the observed files were all in “c:\ProgramData”.

The table above provides one small piece of additional evidence that the same operating group is responsible: the presence of files uploaded named “readme” each leading to a different malware payload. This, along with the fact that these deployment techniques emerged in close temporal proximity, lends more evidence to the “common operator” theory, although there are still limitations regarding how far this assessment can go.

This blog also notes that not every TinyPOS sample is identical in code and configuration. Specifically, the t.bin sample analyzed that matches the Carbon Black/VISA reports and the ccv_server.txt file described in a reddit post appear to target specific processes, with evidence from the latter source suggesting these may be specific to the target environments. Other samples analyzed in the “pivoting” section tagged as “TinyPOS behavior” contain a process blacklist rather than a process target list. This is noted primarily for the sake of technical accuracy and completeness, although these characteristics may be explored in a future post.

III.Conclusion

Based on the observed commonalities surrounding these two malware families and how they are used, it seems likely that there is *some* relationship between them. Originally, this research began as an effort to identify threat actor C2 infrastructure related to the incidents described by VISA’s report; unfortunately, that type of infrastructure is precisely what is still needed to more definitively solve this puzzle.

If evidence emerged that a ProLocker attack and a TinyPOS attack each relied on the same infrastructure, this would suggest a workflow in which a threat actor group gains access to a network, triages it, and then chooses the type of attack it wishes to launch. This, in turn, would be consistent with other threat actor groups that operate in this space.