SECURITY (HTTPS://BLOG.GIGAMON.COM/CATEGORY/SECURITY/) / JULY 26, 2017

Footprints of FIN7: Tracking Actor Patterns (Part 2)

ATR

(https://blog.gigamon.com/author/atrteam/)

Justin Warner

(https://blog.gigamon.com/author/justin-warner/)

Stephen Hinck

(https://blog.gigamon.com/author/stephen-hinck/)

This is part two of a blog series detailing Gigamon Applied Threat Research's (ATR) engagements with FIN7 throughout early 2017. Part one (https://blog.gigamon.com/2017/07/25/footprints-of-fin7-tracking-actor-patterns-part-1//) focused the network communications and tradecraft involved with FIN7, specifically addressing how patterns in the C2 protocol allowed for Gigamon ATR to gain a deeper understanding of adversary TTPs. In this

post, we will break out one of the ways in which FIN7 profits from their victims — compromise of point-of-sale (POS) environments and theft of cardholder data.

End Goal: Financial Gain

Based on victimology, internal attacker actions and known data exfiltration, many people have demonstrated FIN7's clear financial motivations (https://www.fireeye.com/blog/threat-research/2017/04/fin7-phishing-lnk.html). With that said, up until recently, little detail has been shared about the tradecraft employed to accomplish that objective. During our engagements, Gigamon ATR observed a standalone utility that appeared to be custom in nature and used specifically to target POS systems.

Multi-Stage Dropper

During post-exploitation, FIN7 would gain access to the POS environment primarily using PSExec (https://technet.microsoft.com/en-us/sysinternals/bb897553.aspx), combined with a custom PowerShell script to deploy their payload. The weaponized PowerShell script was built to load an embedded DLL using reflective injection. While the PowerShell code appeared to include several foundational components from PowerSploit's Invoke-ReflectivePEInjection (https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-ReflectivePEInjection.ps1), it also contained custom modifications that made it unique.

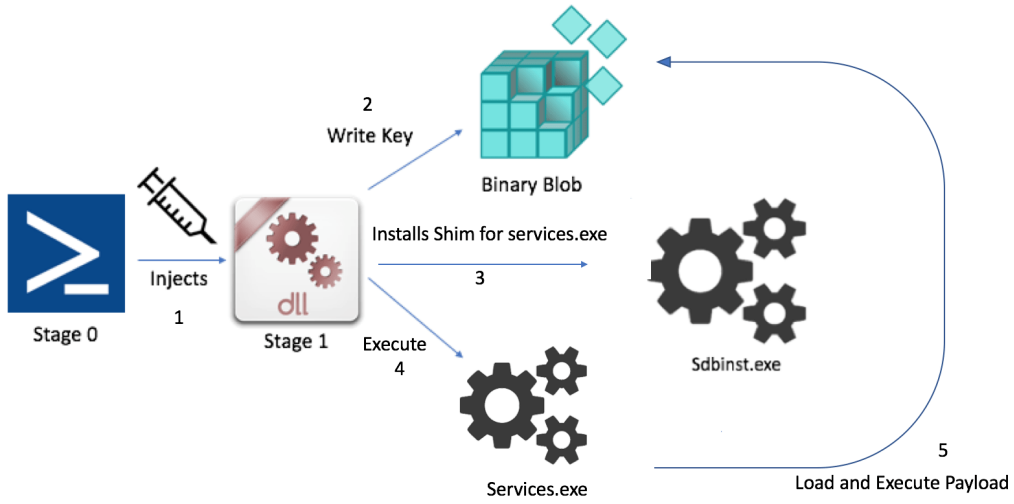
To perform the exploit, the script executed code from PowerSploit designed to resolve Windows API functions necessary for performing the injection. Next, the script would allocate memory in its current process and use the *System.Runtime.Interopservices.Marshal* class to copy code into the newly allocated memory. With the DLL in memory, the script would then generate custom bootstrap code for binary loading and place that code into memory. Finally, the script would create a thread at the offset of the bootstrap code that will in turn load the malicious payload from memory.

```
{
  [Byte[]] $CallStub = 0x68 #push userDataSize
  $CallStub += ConvertTo-LittleEndian $Opts.UserDataSize 4 # userDataSize
  $CallStub += 0x48, 0x83, 0xEC, 0x20 #sub rsp, 0x20
  $CallStub += 0x48, 0xB9 #mov rcx, lpParam
  $CallStub += ConvertTo-LittleEndian $Opts.LpParam # lpParam
  $CallStub += 0x48, 0xBA #mov rdx, &imageBase
  $CallStub += ConvertTo-LittleEndian $Opts.ImageBaseAddr # &imageBase
  $CallStub += 0x41, 0xB8 #mov r8d, &funcHash
  $CallStub += ConvertTo-LittleEndian $Opts.FuncHash 4 # &funcHash
  $CallStub += 0x49, 0xB9 #mov r9, &userData
  $CallStub += ConvertTo-LittleEndian $Opts.UserDataAddr # &userData
  $CallStub += 0x48, 0xB8 #mov rax, &reflectLoader
  $CallStub += ConvertTo-LittleEndian $Opts.ReflectLoaderAddr # &reflectLoader
  $CallStub += 0xFF, 0xD0 #call rax
  $CallStub += 0x48, 0x89, 0xC1 #mov rcx, rax (return code from ReflectiveLoader)
  $CallStub += 0x48, 0xB8 #mov rax, &exitThread
  $CallStub += ConvertTo-LittleEndian $Opts.ExitThreadAddr # &exitThread
  $CallStub += 0xFF, 0xD0 #call rax
}
```

(https://blog.gigamon.com/wp-content/uploads/2020/06/1-7-25-Figure-9-ex-of-bootstrap-generation-code.png)

Figure 9: An example of the bootstrap generation code.

Once injected, the payload proceeded to write a binary encoded blob into the registry key *HKLM\Software\Microsoft\DRM* and install persistence using a custom Shim targeting *services.exe* and adding it to the shim database by executing *sdbinst.exe* (https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html). When *services.exe* is executed on boot, the blob, which is primarily composed of a POS scraping capability, is loaded.



(https://blog.gigamon.com/wp-content/uploads/2020/06/2-7-25-Figure-10-Multistage-Loading-Process.png)

Figure 10: Multistage loading process.

Based on previous public documentation of FIN7 operations (https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html), this custom PowerShell loader is a notable divergence from previous FIN7 operational tactics (https://www.gigamon.com/content/gigamon/en_us.html) (https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html); a shift in techniques possibly to evade detection based on intel reporting.

Notably, there is an inherent risk of detection that an adversary must evaluate when deploying a persistence mechanism, especially in POS environments that experience relatively little day-to-day changes. This shift identifies a likely desire to maintain longer-term access in order to facilitate ongoing theft from a single target environment over time, rather than performing smash and grab campaigns.

Scraper Payload

Once running, the POS scraping capability will attempt to inject itself into *svchost.exe*, *wuauclt.exe* or *explorer.exe* (in that order). Upon successful injection, the scraper enumerates running processes, comparing them to a list provided via a text file on disk. This tactic of using a targeted set of processes allows the malware to be used flexibly in different environments and shows customization in targeting based on each individual victim.

Once the scraping capability has identified a process of interest, the payload will read memory from the process and extract track one and track two data formats. Once extracted, the malware encrypts the data and writes it to disk for later collection by the threat actors.

TOP

Point-of-Sale Considerations

While working with customers of all different sizes and across industries, Gigamon ATR has seen a large mix of architectures that have had both successes and failures, largely as a result of decisions made during the design process. We work with our customers to understand how their individual environment design affects their ability to perform network monitoring, but by the time we are brought in to assist in response it's too late and the priority is containment, mitigation and remediation with the topic of design being revisited in the wake of the breach. With that said, we hope to present some considerations that apply specifically to protecting and monitoring POS environments:

- **Network isolation:** POS systems and processing servers should be relatively isolated, with strict access controls

applied in the environment. In most POS environments, remote networks (such as those found in storefronts) have no need to connect directly to each other, instead network communication should exist only between the remote networks, and any centralized processing or storage resources. Where possible, Gigamon ATR recommends the application of network controls that enforce this model.

- **Monitoring coverage:** Wherever possible, Gigamon ATR recommends backhauling all internet traffic through centralized points of presence for ease of consistent monitoring. On multiple occasions, Gigamon ATR has witnessed the use of split tunneling (sending local traffic directly to the internet) create difficulties in monitoring that traffic. This hinders or prevents the ability to detect malware communicating external to the environment. Proxying and logging all network communications from POS systems, preferably in a robust security stack that allows for behavioral profiling, enables the detection of, and response to, threats in the environment.
- **Behavioral monitoring:** Network traffic behavior in POS environments should be fairly predictable, and deviations from that behavior should warrant additional investigation. By monitoring and baselining network flows within the POS environment, defenders can identify changes to those patterns. Some examples of patterns to watch include: network traffic volume, hosts communicating with each other, the direction of that communication and the amount of data transmitted.
- **Application whitelisting:** Similar to network behaviors, the applications running in POS environments remain fairly

static. Baselining the environment, and then applying controls that allow only a small set of required applications to execute on hosts involved in processing payment data, can prevent attackers from ever gaining a foothold on those systems.

- **Host-based firewalls:** In addition to whitelisting applications allowed to execute on system involved in processing payment data, whitelisting the ports and IPs that each system is allowed to communicate on will help to prevent lateral movement within the POS environment. Controlling network traffic, particularly on ports used for system management (commonly used for lateral movement), ensures that such activity can only occur from certain hosts, and can greatly increase (if not prevent completely, in combination with other controls) the amount of time it takes to pivot within the POS environment, increasing the likelihood of discovery of an active attacker.

Conclusion

The information-security industry as a whole commonly puts a lot of focus on top tier threats such as nation-state actors. While very real, these threats are a part of a different model than most organizations should likely consider. There are serious threats facing the average business and these groups continue to elevate their sophistication and learn from their engagements. With one goal in mind, these actors will continue to attack environments in an effort to profit off of the blindness of their victims.

While FIN7 is a moderately sophisticated threat, the patterns of their operations and tradecraft decisions can be reliably detected by anyone with good visibility into their environment. Further, early investments in security-focused design and realistic threat modeling will allow organizations to better prepare for the inevitable targeting by one of these groups. We hope that these posts were insightful and allow you to bring similar defensive tradecraft into your environment.

Thanks for reading! To learn more about Gigamon ATR, please visit www.gigamon.com/research/applied-threat-research-team.html (https://www.gigamon.com/research/applied-threat-research-team.html).

OLDER ARTICLE

Are You Effectively Securing Your Cloud Workloads? Learn How Gigamon and RSA Can Help (https://blog.gigamon.com/2017/07/25/effectively-securing-cloud-workloads-learn-gigamon-rsa-can-help/)

NEWER ARTICLE

Footprints of Fin7: Tracking Actor Patterns (Part 1) (https://blog.gigamon.com/2017/07/26/footprints-of-fin7-tracking-actor-patterns-part-1/)