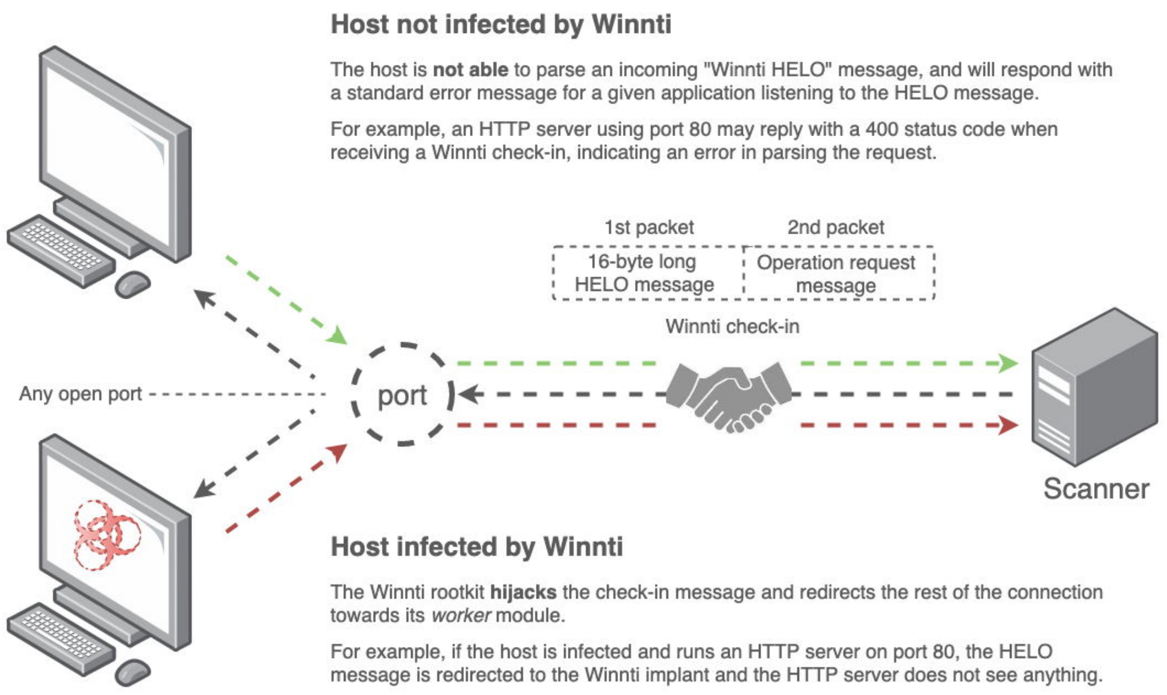
HELO Winnti: Attack or Scan?

lastline.com/labsblog/helo-winnti-attack-scan/

September 30, 2019


Host not infected by Winnti
The host is **not able** to parse an incoming "Winnti HELO" message, and will respond with a standard error message for a given application listening to the HELO message.
For example, an HTTP server using port 80 may reply with a 400 status code when receiving a Winnti check-in, indicating an error in parsing the request.
1st packet
2nd packet
16-byte long HELO message
Operation request message
Winnti check-in
Any open port
port
Scanner
Host infected by Winnti
The Winnti rootkit **hijacks** the check-in message and redirects the rest of the connection towards its *worker* module.
For example, if the host is infected and runs an HTTP server on port 80, the HELO message is redirected to the Winnti implant and the HTTP server does not see anything.

Posted by Jason Zhang and Stefano Ortolani ON SEP 30, 2019
Since its first attack was discovered nearly a decade ago, Winnti has evolved into an advanced and sophisticated toolkit leveraged by several actors such as APT17, Axiom, Barium, and PassCV, just to name a few. All these actors have been sharing core tactics, techniques, and procedures (TTPs), leading to highly persistent implants targeting organizations from the online gaming industry to high-tech companies around the globe. It comes as no surprise that both security vendors and researchers are still scrutinizing and tracking this specific threat.

While no new campaign has been publicly reported after the one targeting German Pharmaceutical companies in April 2019 [1], since July 2019 we have been seeing a dramatic increase of network activity related to Winnti. Figure 1 displays all daily alerts from more than two months of telemetry data since mid-June 2019.

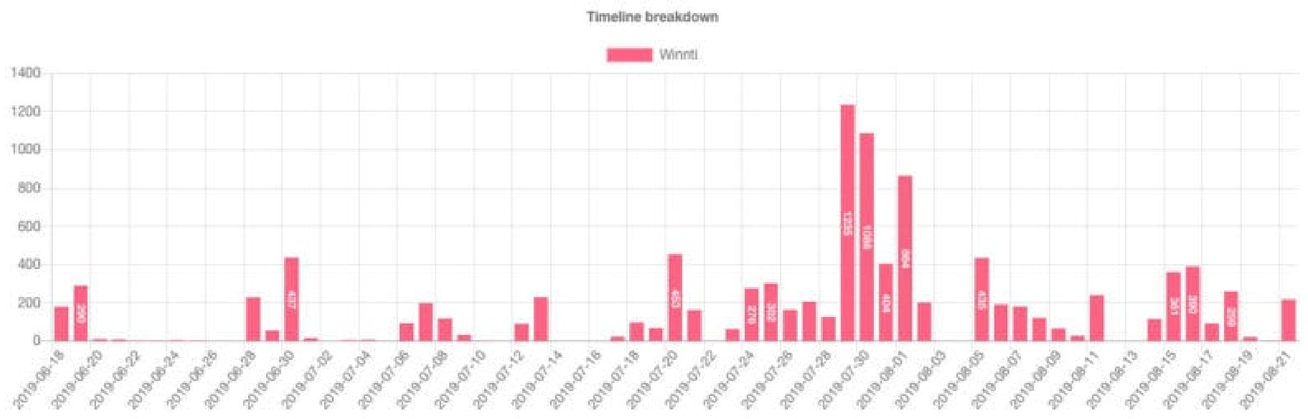


Figure 1: Winnti daily alerts timeline.

As it turned out, this telemetered traffic was due to external Winnti check-ins or “Winnti HELO” messages sent out by non-malicious Winnti scanners looking for hosts infected by the Winnti implant. Thanks to the massive increase of investigation-oriented traffic, the actual signal from real attacks is buried in the noise generated by scan traffic. Such extremely low signal-to-noise ratio poses huge challenges for both investigators and customers looking to identify real attacks.

In this report, we investigate the magnitude of this phenomenon, attempt to mitigate these challenges, and propose an effective triage process benefiting the whole security community. We start by providing a brief overview of the Winnti toolkit implants lifecycle. Then, we discuss the last two months of internal telemetry data showing the dramatic increase of Winnti traffic from various sources, and explain why this is a challenging problem for both researchers and network administrators. Finally, we provide a deep dive on the issue, share our findings, and propose an effective triage process tailored to both researchers and security professionals.

What is Winnti?

Winnti represents a malware family with Remote Access Trojan (RAT) functionalities. The memory-resident malware is highly persistent: once it is successfully installed on a victim’s host, it gives the criminals the capability to control the infected computer without the victim’s knowledge. Historically, the targets attacked vary from online gaming companies to pharmaceutical industry (more details are discussed in a later section).

The actors behind Winnti have been active at least since 2009, and, throughout the decade since its initial attack, Winnit has been evolving into a more advanced and sophisticated toolkit. The toolkit comprises four main components, all taking part in the execution cycle (as thoroughly explained by Novetta in their Winnti Analysis [3]):

- *Dropper*: a module responsible for dropping the Winnti malware on a victim’s machine
- *Worker*: a module responsible for communication (such as parsing HELO messages) and plugin management
- *Service*: a module whose primary function is to activate the engine component
- *Engine*: a module that only exists in memory once activated by the *Service* component; its main function is to fulfill the malware installation process after the *Service* component passes control to the Install function in the engine component

Extensive investigations on the malware have been published by Kaspersky [2], Novetta [3], and Thyssenkrupp's CERT [4], with a more recent case study by Chronicle [5] about a Linux variant. Once the highly persistent memory-resident implant is successfully installed on a victim's host, the malware operators can initiate a check-in connection directly to an infected host without relying on a C2 server (though it does call out to a C2 server to receive commands in some cases). Figure 2 shows the main timeline of the malware development since the first reported attack and the possible threat groups behind the attacks according to various reports [2-3, 6].

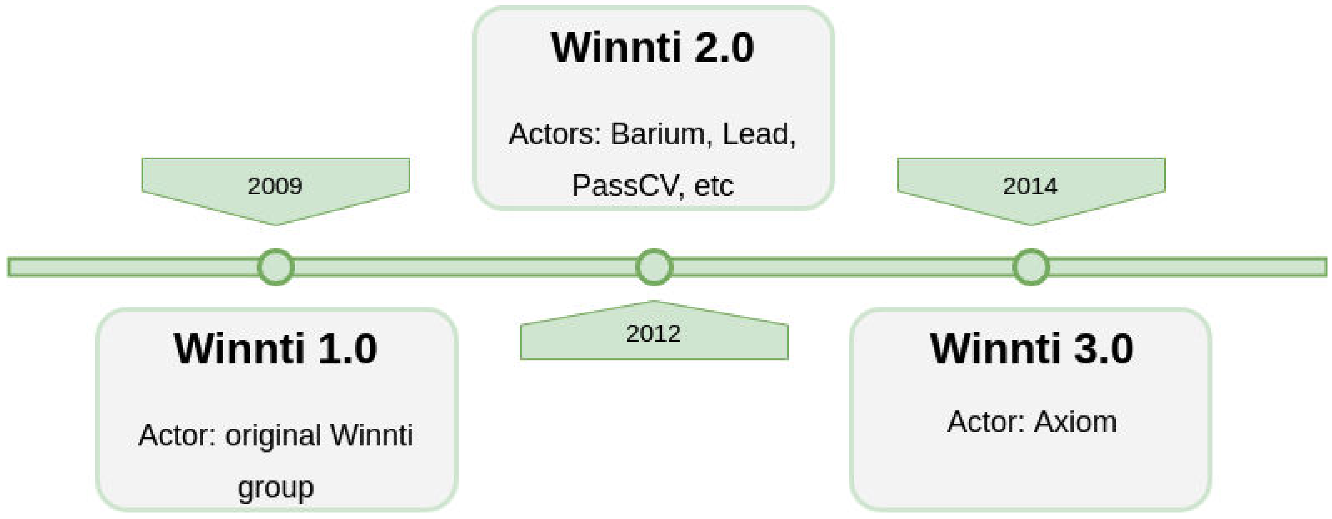

Winnti 2.0
Actors: Barium, Lead, PassCV, etc
2009
2014
2012
Winnti 1.0
Actor: original Winnti group
Winnti 3.0
Actor: Axiom

Figure 2: Winnti evolution timeline.

Winnti 1.0/2.0

Kasperksy initially reported on attacks relying on Winnti 1.0 and 2.0 in 2013, including evidence of activity dating back to 2009 [2]. Winnti 1.0 was capable of stealing certificates from its targets, and is believed to be the very first identified malware with a valid (stolen) certificate targeting the Windows 64-bit operating systems. Certificates have been becoming increasingly important as recent operating systems have been starting to limit execution to digitally signed executables. As a consequence, the abuse of digitally signed malware became a very effective and sustainable approach in Winnti-related attacks. The APT17, Barium, Lead, and PassCV groups are thought to have strong connections with these attacks [6].

Winnti 3.0

Winnti 3.0 was first made public in the article published by Novetta in 2015 [3]. The report notes that the malware was actually compiled in mid-to-late 2014. According to the Novetta's findings, Winnti 3.0 is very similar to its predecessors, but the dropper component evolved to incorporate some anti-analysis mechanisms, making the malware investigation more difficult. The criminal activity exploiting Winnti 3.0 exhibits TTPs that are very similar to attacks operated by the Axiom group, which is known to carry out cyber-espionage attacks against a whole range of industries.

Targeted Industries: From Games to Pills

Winnti-based attacks initially targeted online gaming companies. The motivation for these attacks was twofold. Once gained control of the game servers, criminals could monetize the attack by converting virtual funds into real money and/or deploying stolen source code from the compromised servers into their own pirate servers. The second objective was to steal code-signing certificates, which could then be used to sign malicious applications when attacking higher-value organizations.

Such exclusive targeting (only game organizations) changed by the end of 2014, marking the expansion of the target focus to other industries, German chemical and pharmaceutical conglomerates in particular [2-3]. The latest victims of this wave were reported in April 2019 [1]. Note that the list of targets does not stop here, as political activists and news portals had also been targeted [2,6]. Figure 3 summarizes which entities have been targeted over time, based on the information we collected at the time of the investigation.

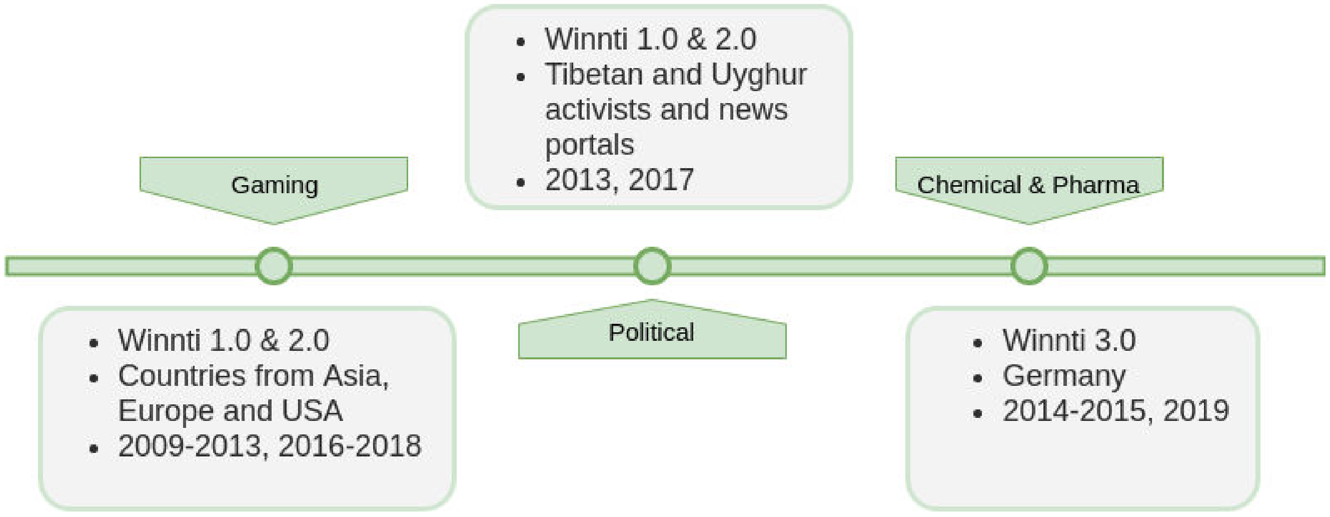


Figure 3: Winnti targeted entities over time.

Tracking Winnti

As we mentioned before, once the implant is installed on an infected host, the attack operators can directly communicate with the host [5]. To be more specific, a rootkit deployed together with the *worker* component allows Winnti to intercept and redirect specifically crafted TCP connections from ports already used by legitimate applications: if the first part of the payload of the TCP flow is 16 bytes long, and follows a specific format, the rootkit redirects the rest of the connection towards its *worker* module (see [3,5] for more details).

This specific communication approach between the threat actor and the infected host provides a great way for investigators to track Winnti activity and subsequently detect the malware. This was successfully demonstrated by ThyssenKrupp's CERT, as they recently released a set of open-source tools able to detect and mimic "Winnti HELO" messages [4]. The underlying logic is quite straightforward: the tool starts by sending a 16-byte long message to a candidate host (any destination port already open would work), and checks the replied message to determine whether the implant is installed. This allowed security companies and professionals to develop and deploy mass-scan agents scouting the Internet, hoping to find hosts infected by the Winnti implant.

Therefore, determining whether a host is infected is equivalent to finding out how the host responds to a “Winnit HELO” message (first packet of the Winnti check-in). This leads to two different scenarios (see Figure 4):

- If a host is **not** infected by Winnti, it will just reply with an error message depending on the particular application listening on the port receiving the “Winnti HELO” packet.
- If instead the host **is infected**, the Winnti rootkit will hijack the “Winnti HELO” message, and redirect it to the *worker* module of the malware. Once the HELO message is validated, the *worker* component checks if the 3rd DWORD value in the operation request message (second packet of the Winnti check-in) is **\0xABC18CBA**, the magic number [5]. If verified successfully, it will execute the request and provide the host’s system information.

It is worth noting that the “Winnti HELO” message (the first packet of a Winnti check-in as shown in Figure 4) comprises 4 DWORDs generated by a pseudo-random integer number generator [4-5]. We will detail how the HELO messages are generated in a later section.

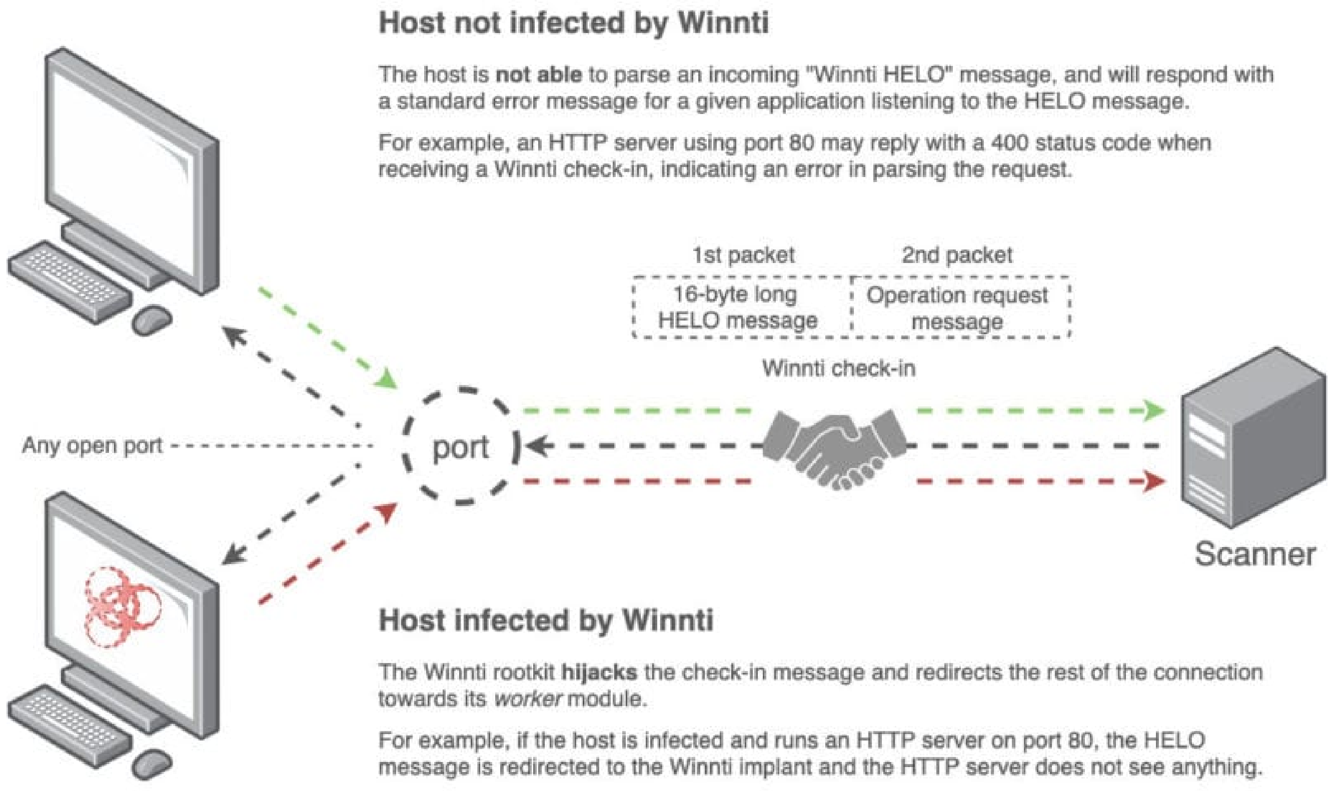

Host not infected by Winnti
The host is not able to parse an incoming "Winnti HELO" message, and will respond with a standard error message for a given application listening to the HELO message.
For example, an HTTP server using port 80 may reply with a 400 status code when receiving a Winnti check-in, indicating an error in parsing the request.
1st packet
16-byte long HELO message
2nd packet
Operation request message
Winnti check-in
Any open port
port
Scanner
Host infected by Winnti
The Winnti rootkit hijacks the check-in message and redirects the rest of the connection towards its worker module.
For example, if the host is infected and runs an HTTP server on port 80, the HELO message is redirected to the Winnti implant and the HTTP server does not see anything.

Figure 4: Winnti check-ins and responses from hosts with/out infection.

New challenges

The idea of using mimicked “Winnti HELO” messages does indeed ease the challenge of finding infected hosts, but it also introduces a new problem: noise. Since all publicly available signatures (like the Suricata IDS rule released as part of the aforementioned open-source tools) flag a possible Winnti attack whenever a “Winnti HELO” message is detected, each external scan now automatically translates to a new alert.

As a result, the superimposed network traffic with dominant noisy scan signals poses huge challenges for both researchers and SOC analysts. In particular:

- It greatly impairs the threat hunters' ability to monitor the threat actors' activity. Although emulating the "Winnti HELO" message does help to disclose the number of implants that are present in a network, the process of differentiating real attacks from benign scans becomes much more complicated. Before researchers started scanning the whole Internet, every rogue HELO message would have been correlated to an actual attack. Now, it is no longer the case.
- It also significantly slows down the triage of a network event in a SOC, which is already challenged by a non-negligible number of cyber alerts and attacks from other vectors. The dramatic increase of alerts caused by non-malicious Winnti scanners requires more resources and effort for investigation, making the analysts' work even more challenging and less efficient.

As it is often the case, the situation can be eased by deploying more complex but expensive solutions, able to deep-inspect the Winnti protocol. This approach might not however be practical for many organizations, small companies in particular, which are left with several challenges to effectively triage Winnti alerts. In the following sections, we will briefly review the most common examples, and we will conclude with an effective and easy-to-implement method that can considerably mitigate all these challenges.

Rely on source IP addresses? No

One might suggest to use the source IP address to determine whether the check-in is a scan or an attack. Figure 5 illustrates what a Winnti check-in traffic looks like, where the host *192.168.2.6*, part of the protected network, receives check-in messages from both the *218.211.168.178* and *155.94.254.143* IP addresses. Is there an attack within the traffic?

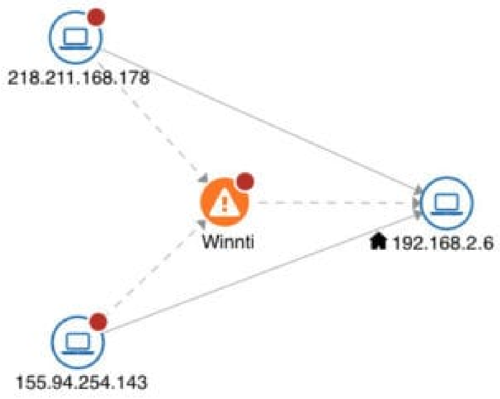


Figure 5: Is it an attack or a scan?

There is no easy answer to this question, because the scans are generated from several source IP addresses, many changing every week, making whitelisting a non-scalable work-around. Figure 6 shows the source IP address distribution based on two months of internal telemetry data since June 2019.

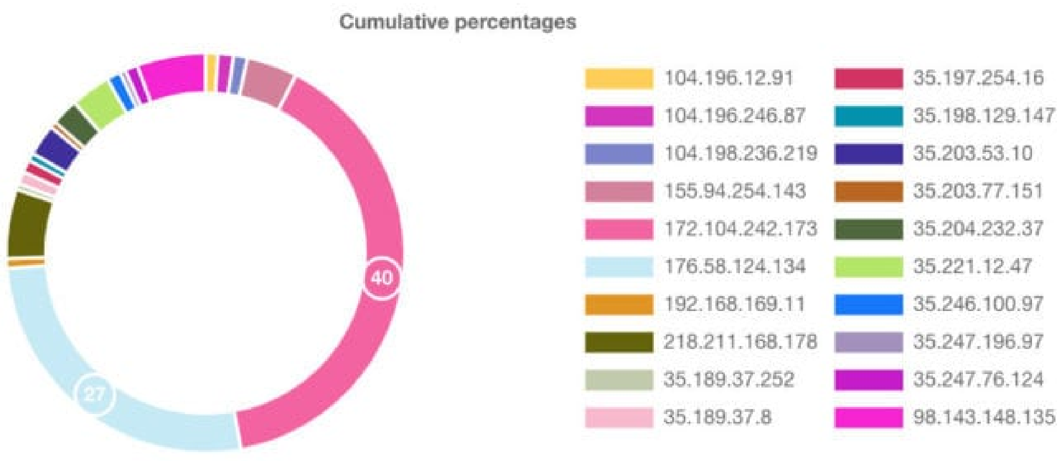
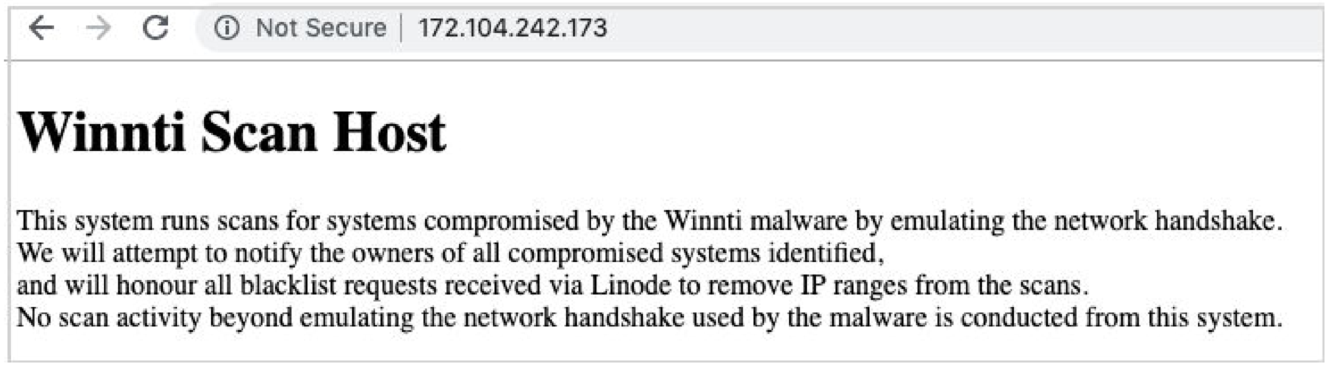


Figure 6: Check-in source IP address distribution between June – August 2019.

As we can see, the IP addresses are from a wide range of sources. Interestingly, some are well-known scanners (which can be easily whitelisted), but many IP addresses are unknown, or belong to random cloud VMs (possibly provisioned/deployed by researchers inspired by recent news). Examples of well-known scanners are:

- 155.94.254.143: scanner03.project25499.com
- 172.104.242.173: winnti-scanner-victims-will-be-notified.threatsinkhole.com
- 176.58.124.134: tequilaboomboom.club
- 98.143.148.135: scanner05.project25499.com

The two IP addresses dominating the chart above are from this last set. For instance, one has a single self-explanatory page with the following message:



Winnti Scan Host

This system runs scans for systems compromised by the Winnti malware by emulating the network handshake.
We will attempt to notify the owners of all compromised systems identified,
and will honour all blacklist requests received via Linode to remove IP ranges from the scans.
No scan activity beyond emulating the network handshake used by the malware is conducted from this system.

Figure 7: A landing page from a cloud VM-based scanner.

It is worth noting that we also found some evidence of scans originating from private IP addresses, indicating that some of the check-ins were coming from within the local network. While we acknowledge that proactive security teams might be actively looking for infected endpoints, the most probable explanation is that such internal hosts might also be acting as a router, SNATing connections originating from outside (this normally happens when the traffic is not monitored from the perimeter but from inside the network), as depicted in Figure 8.

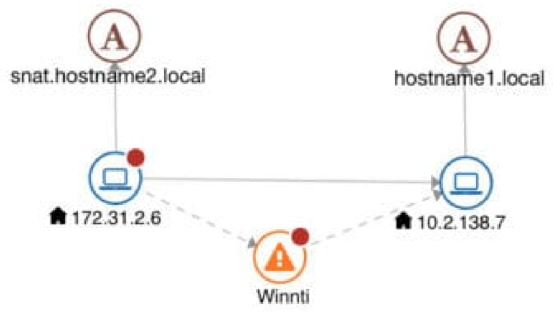


Figure 8: A Winnti scans appearing to originate from a private IP address; the most likely explanation being an external host contacting a service hosted on 10.2.138.7, whose port was forwarded to the DMZ by the router running on 172.31.2.6.

Admittedly, this can be very confusing for SOC analysts: the mere presence of a check-in originating from inside the perimeter is an obvious cause of concern. It is clear how the investigation into the causes can take a significant amount of time and effort. This demonstrates that simply whitelisting private IP addresses without proper investigation may lead to security risks.

Rely on destination ports? No

Also destination ports are not reliable, simply because they are meaningless for Winnti, as it can basically use any open port. The chart below shows the distribution of scan destination ports in our internal telemetry data.

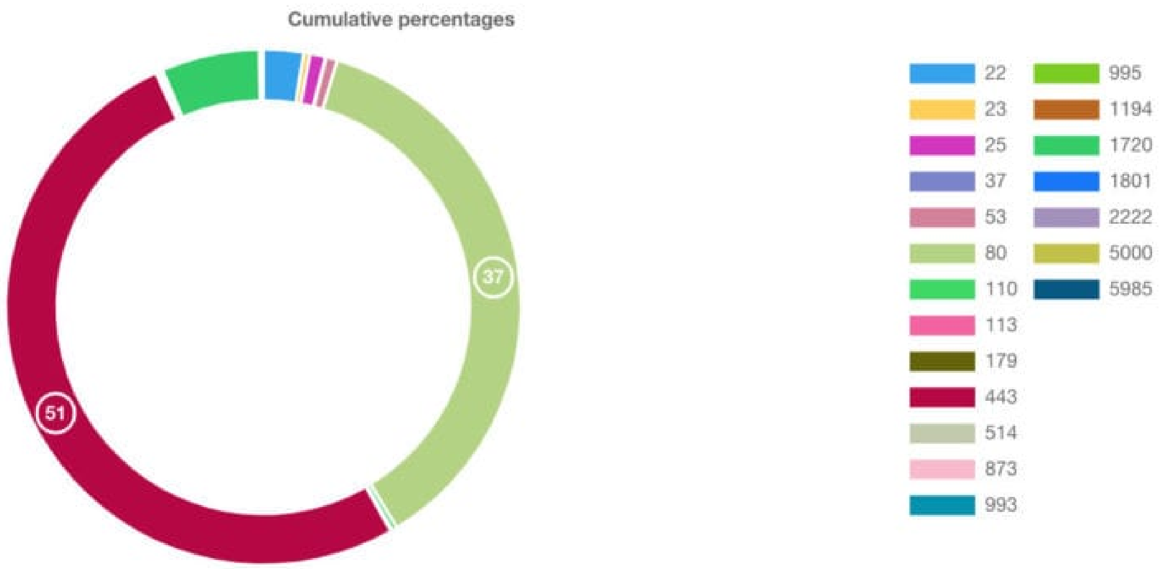


Figure 9: Scan destination port distribution (percentage) between June – August 2019.

As Figure 9 shows, the scans are distributed over a number of ports. The reason why we see more check-ins on 80 and 443 (HTTP/HTTPs) ports is because those are normally available to all connections, internal and external alike (unlike, say, SSH).

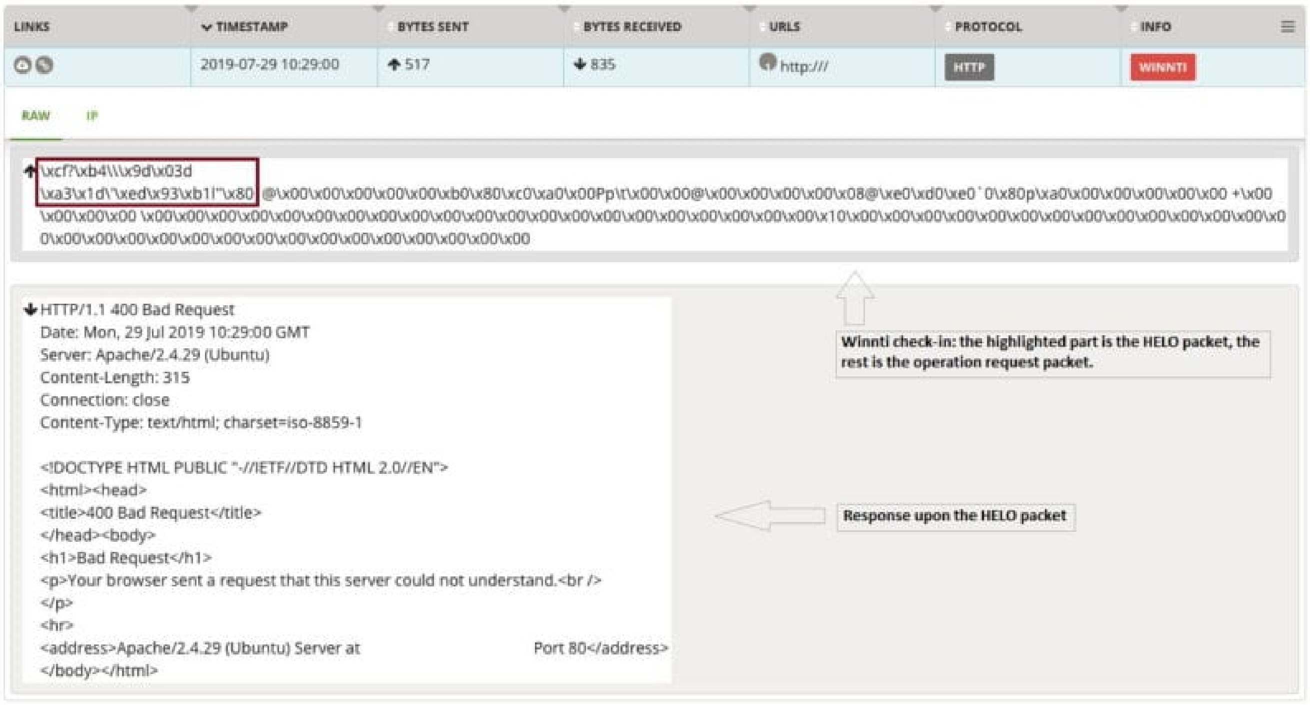
Mitigation

A successful three-way handshake is the first requirement for a check-in to be successful (regardless of the fact that it was a scan or a real attack). If the connection is not successful then we know there is no infection. The only case where further analysis is required is when there is an established connection, and in such case we need to inspect the response by taking into account the specific application-layer protocol used by the server to reply.

As pointed out earlier, if the first message of the TCP flow is 16 bytes long (and it follows a specific format), the rootkit will redirect the rest of the connection towards its *worker* module. This connection hijacking capability allows us to effectively identify whether a host is infected or not. In other words, if the application-layer protocol returns a standard message upon a “Winnti HELO” check-in, it implies that the host is not infected (the check-in did not result in a hijacked connection). More details are discussed in the following typical cases.

HTTP

When the attackers send the Winnti HELO message using a connection established on port 80, the most likely service listening, if any, is a web server. This means that if the request can not be parsed by the userspace process (the web server), the most likely answer is a web page detailing that the parsing failed. As the screenshot below shows, the HTTP server returns a 400 status page upon the incoming “Winnti HELO” check-in. This indicates that the server is not infected by Winnti.


LINKS	TIMESTAMP	BYTES SENT	BYTES RECEIVED	URLS	PROTOCOL	INFO
	2019-07-29 10:29:00	517	835	http:///	HTTP	WINNTI

```
\xcf?\xb4\\\x9d\x03d
\xa3\x1d\"\xed\x93\xb1l"\x80 @\x00\x00\x00\x00\x00\xb0\x80\xc0\xa0\x00Pp\t\x00\x00@\x00\x00\x00\x00\x08@\xe0\xd0\xe0`0\x80p\xa0\x00\x00\x00\x00\x00 +\x00
\x00\x00\x00 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

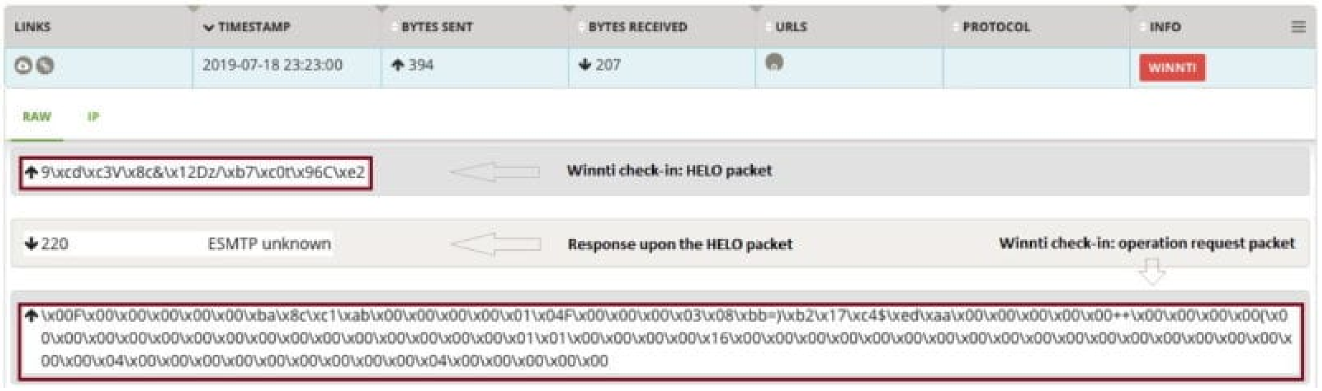
```
HTTP/1.1 400 Bad Request
Date: Mon, 29 Jul 2019 10:29:00 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 315
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.29 (Ubuntu) Server at                    Port 80</address>
</body></html>
```

Winnti check-in: the highlighted part is the HELO packet, the rest is the operation request packet.
Response upon the HELO packet

Figure 10: An HTTP server returns a parsing error in response to a “Winnti HELO” check-in.

SMTP

Similarly, if an SMTP server being scanned returns an error message after receiving a Winnti check-in on port 25 (most likely), it is clear that the connection reached the user space application. This implies that there was no Winnti implant on the host. The figure below illustrates the inbound and outbound messages on an SMTP server. The server received two incoming packets from a Winnti connection. The first one is a 16-byte long “Winnti HELO” message, followed by an encrypted message. As the SMTP server is not infected, it responded with an error message, indicating that it was not able to parse the request from the first packet.



LINKS	TIMESTAMP	BYTES SENT	BYTES RECEIVED	URLS	PROTOCOL	INFO
	2019-07-18 23:23:00	394	207			WINNTI

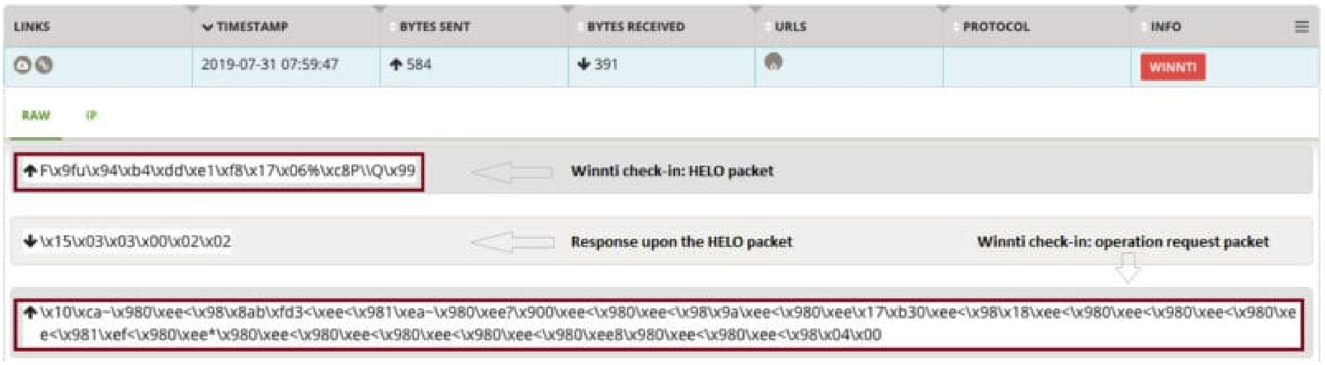
```
9\xcd\xc3V\x8c&\x12Dz/\xb7\xc0t\x96C\xe2          Winnti check-in: HELO packet

220          ESMTP unknown          Response upon the HELO packet          Winnti check-in: operation request packet

\x00F\x00\x00\x00\x00\x00\xba\x8c\xc1\xab\x00\x00\x00\x00\x01\x04F\x00\x00\x00\x03\x08\xbb=)\xb2\x17\xc4$\xed\xaa\x00\x00\x00\x00\x00++\x00\x00\x00\x00(\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x16\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00
```

Figure 11: An SMTP server replies with a server message upon “Winnti HELO” check-in.

It is worth noting that all communications using other protocols such as Telnet or FTP (clear-text protocols) can be triaged by inspecting the protocols. Most of the time it is easy to understand whether the response is legitimate, albeit the process can be time-consuming

HTTPS

Checking TLS-encrypted connections is much more challenging, because the data received from an endpoint does not clearly show whether the response is generated by a Winnti implant or just by the associated server.



LINKS	TIMESTAMP	BYTES SENT	BYTES RECEIVED	URLS	PROTOCOL	INFO
	2019-07-31 07:59:47	584	391			WINNTI

```
F\x9fu\x94\xb4\xdd\xe1\xf8\x17\x06%\xc8P\\Q\x99          Winnti check-in: HELO packet

\x15\x03\x03\x00\x02\x02          Response upon the HELO packet          Winnti check-in: operation request packet

\x10\xca~\x980\xee<\x98\x8ab\xfd3<\xee<\x981\xea~\x980\xee?\x900\xee<\x980\xee<\x98\x9a\xee<\x980\xee\x17\xb30\xee<\x98\x18\xee<\x980\xee<\x980\xee<\x980\xe
e<\x981\xef<\x980\xee*\x980\xee<\x980\xee<\x980\xee<\x980\xee<\x980\xee8\x980\xee<\x980\xee<\x98\x04\x00
```

Figure 12: An HTTPS server replies a short message upon “Winnti HELO” check-in.

In these cases, we need to rely on other clues, for example the intrinsic characteristics of a valid response to a Winnti check-in. The Winnti underlying protocol relies on responses that are 16 bytes long. This is because the initial handshake needs at least 16 bytes to encode details of the key exchange mechanism. This means that if the encrypted response to a Winnti HELO packet is shorter than that amount, then we can infer that the host is not infected. Coincidentally, the TLS record sent when the request can not be parsed is smaller (as shown in Figure 12, more details on TLS handshake protocol can be found in [8]). Our statistics show that over 50% of all check-ins are on port 443 (HTTPs). All of them are less than 16 bytes. Hence more than half of all the received check-ins can be ruled out deterministically without requiring manual inspection.

HELO values distribution

While analyzing the traffic, we also turned our attention to the DWORDs (HELO values from now on) used to assemble the first packet of the Winnti check-in and found some anomalies about the value distribution. Interestingly, we identified some artifacts in a popular scanner [4] that would often lead to the generation of identical HELO values, devoid of any randomization required to keep the generated traffic indistinguishable from malicious HELO messages. Case in point, we found the following HELO message to be one of the most prevalent:

```
9\xcd\xc3V\x8c&\x12Dz/\xb7\xc0t\x96C\xe2
```

Based on our telemetry data, these specific HELO values had been used 6,429 times out of all 14,962 check-ins we collected, which is about 43% of all the records.

Another interesting anomaly is related to the message detailing the operation requested (the byte sequence immediately following the HELO values): 32.4% of the time, for unknown reasons (but we speculate that this might be caused by some errors in the scanner logic), it lacked encryption (you can in fact notice the magic values **\0xABC18CBA** highlighted below should be encrypted in the operation request, as often documented in other articles [4-5]).

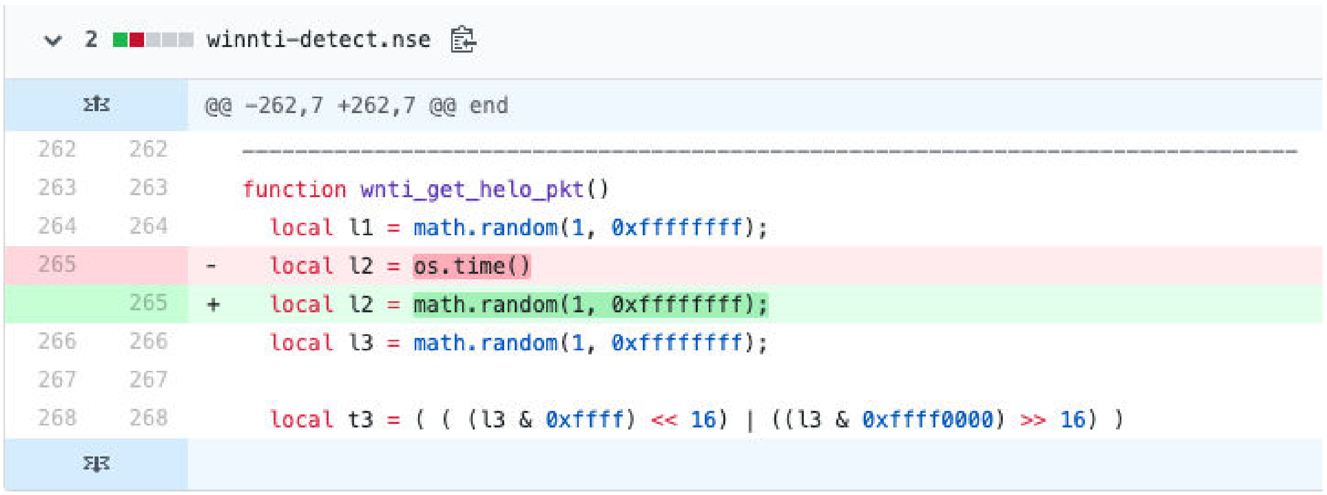
```
\x00F\x00\x00\x00\x00\x00\xba\x8c\xc1\xab\x00\x00\x00\x00\x01\x04F\x00\x00\x00\x03
\x08\xbb=)\xb2\x17\xc4$\xed\xaa\x00\x00\x00\x00\x00++\x00\x00\x00\x00(\x00\x00\x00
```

We also identified an additional very frequent HELO message:

```
\xae\xd0\xab,>\xf9JB2.r\xc2\xd9\xee\x9c\xfe
```

This specific instance appeared 3,591 times (or 24% of 14,962), and 99.999% of the time was followed by the same encrypted operation request sequence.

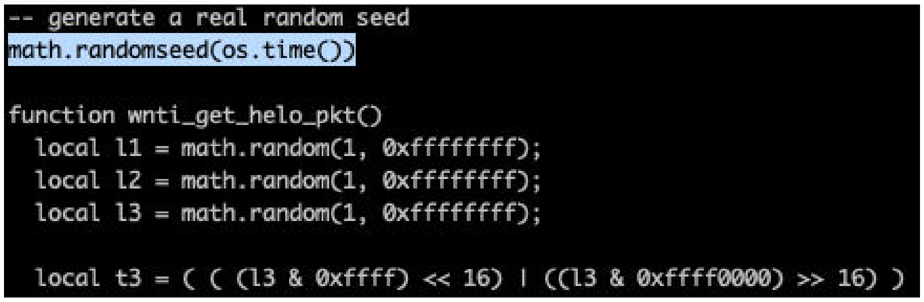
Based on our research the underlying reason why we see so many anomalies generated by the Nmap-based scanner for Winnti, is because the core of the Nmap Scripting Engine is an embeddable Lua interpreter (a lightweight language designed for extensibility). There is a known issue with Lua's internal pseudo-random integer number generator, i.e., `math.random()`, in MacOS and FreeBSD [7]: the difference of the seeds generated when executing the engine on those operating systems is very small, and it gets lost during the random number generation [7]. This explains why the seed remains the same each time `math.random()` is called, resulting in the generated numbers to follow the same exact distribution. More interestingly, this behavior is even more pronounced after the last code commit [4] (see Figure 13).



```
winnti-detect.nse
@@ -262,7 +262,7 @@ end
  --------------------------------------------------------------------------------
  function wnti_get_helo_pkt()
    local l1 = math.random(1, 0xffffffff);
-   local l2 = os.time()
+   local l2 = math.random(1, 0xffffffff);
    local l3 = math.random(1, 0xffffffff);

    local t3 = ( ( (l3 & 0xffff) << 16) | ((l3 & 0xffff0000) >> 16) )
```

Figure 13: Last code change from the GitHub repository [4].

As we can see, `os.time()` was replaced by `math.random(1, 0xffffffff)`. While, before the merge, one of the HELO DWORDs, albeit not random, was still following a monotonic distribution (`os.time()` returns the system time in seconds), after the commit all generated numbers were susceptible to the seed issue we just described, leading to a portion of the HELO message being often constant. This is exactly what we noticed while analyzing Winnti check-ins, confirming our suspicion that such HELO messages were after all generated by benign scanners, not real attacks. It is important to notice that this flaw can also be abused by Winnti actors to selectively discard scan HELO messages. To eliminate this concern, the fix is to add `math.randomseed(os.time()`) (to have a real random seed each time) before calling `math.random()` to get better pseudo-random numbers (see Figure 14).



```
-- generate a real random seed
math.randomseed(os.time())

function wnti_get_helo_pkt()
  local l1 = math.random(1, 0xffffffff);
  local l2 = math.random(1, 0xffffffff);
  local l3 = math.random(1, 0xffffffff);

  local t3 = ( ( (l3 & 0xffff) << 16) | ((l3 & 0xffff0000) >> 16) )
```

Figure 14: Add *math.randomseed(os.time())* to generate pseudo-random numbers.

Note that once the generation of random numbers is fixed as outlined, it is not possible anymore to whitelist specific Winnti check-in packets known to originate from benign scanners.

Conclusions

Over the last decade Winnti has evolved into an advanced and sophisticated toolkit. Nowadays, many organizations of different size still suffer from such attacks. In the meantime, emulated “Winnti HELO” check-ins hoping to identify infected hosts have led to a dramatic increase in network activity, posing new challenges to security professionals whose job is the triaging of network-based attacks. In this report, we carried out a comprehensive investigation of all Winnti alerts collected over two months of internal telemetry data, including:

- Overview of the Winnti toolkit implants life cycle
- The cause behind the dramatic increase of Winnti activity

- New challenges tracking Winnti activity and how they affect SOC's triage processes

Then, we demonstrated how we can considerably mitigate the challenges in differentiating real attacks from benign scans. More specifically,

- If a host is not infected by Winnti, it is not able to parse an incoming "Winnti HELO" message; if there is an application listening on that specific port it will also respond with an application-layer error message.
- If a host is infected by Winnti, the Winnti rootkit hijacks the check-in message and redirects the rest of the connection towards its *worker* module.

In addition, we also investigated the distribution of HELO values and identified a few flaws (e.g., constant values), leading us to believe that such HELO messages were probably generated by benign scanners, not real attacks. Note that a motivated attacker can always modify the current Winnti implant to selectively ignore such "anomalous" check-ins, as a covert attempt to fly under the radar. The fix we propose in this article generates pseudo-random numbers even across different scanner's executions, thus removing this glitch which leads to semi-predictable behavior.

References

1. "Bayer contains cyber attack it says bore Chinese hallmarks", https://in.reuters.com/article/bayer-cyber/bayer-says-has-detected-contained-cyber-attack-idINKCN1RG0NF, accessed: 09/2019
2. " 'Winnti' More than just a game", https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/20134508/winnti-more-than-just-a-game-130410.pdf, accessed: 09/2019
3. "WINNTI ANALYSIS", https://www.novetta.com/wp-content/uploads/2015/04/novetta_winntianalysis.pdf, accessed: 09/2019
4. "Nmap Script to scan for Winnti infections", https://github.com/TKCERT/winnti-nmap-script, accessed: 09/2019
5. "Winnti: More than just Windows and Gates", https://medium.com/chronicle-blog/winnti-more-than-just-windows-and-gates-e4f03436031a, accessed: 09/2019
6. "Burning Umbrella: An Intelligence Report on the Winnti Umbrella and Associated State-Sponsored Attackers", https://401trg.com/burning-umbrella/, accessed: 09/2019
7. "Math Library Tutorial", http://lua-users.org/wiki/MathLibraryTutorial, accessed: 09/2019
8. "The Transport Layer Security (TLS) Protocol Version 1.3", https://www.rfc-editor.org/rfc/pdfrfc/rfc8446.txt.pdf, accessed: 09/2019

- About
- Latest Posts

Jason Zhang

Jason Zhang is a senior threat researcher at Lastline. Prior to joining Lastline, Jason worked at Sophos and MessageLabs (then Symantec) specializing in cutting-edge threat research, and ML application in malware detection. Jason is a regular speaker at leading technical conferences including Black Hat and VB. Jason earned his Ph.D. in Signal Processing from King's College London & Cardiff University.

Latest posts by Jason Zhang (see all)

- About
- Latest Posts

Stefano Ortolani

Stefano Ortolani is Director of Threat Intelligence at Lastline. Prior to that he was part of the research team in Kaspersky Lab in charge of fostering operations with CERTs, governments, universities, and law enforcement agencies. Before that he earned his Ph.D. in Computer Science from the VU University Amsterdam.