Threat Analysis: Active C2 Discovery Using Protocol Emulation Part3 (ShadowPad)

By Takahiro Haruyama

Published: 2022-10-27 · Archived: 2026-04-05 20:19:42 UTC

ShadowPad is a modular malware platform privately shared with multiple PRC-linked threat actors since 2015. According to SentinelOne, ShadowPad is highly likely the successor to PlugX. Due to its prevalence in the cyber espionage field, the VMware Threat Analysis Unit (TAU) was motivated to analyze the command and control (C2) protocol to discover active ShadowPad C2s on the Internet.

ShadowPad supports six C2 protocols: TCP, SSL, HTTP, HTTPS, UDP, and DNS. In this research, TAU focuses on TCP/HTTP(S)/UDP protocols as others like SSL and DNS are not likely utilized by the recent ShadowPad samples.

The format and encoding algorithm is different between TCP and HTTP(S)/UDP.

Table 1: Difference in packet format

	TCP	**HTTP(S)/UDP**
Key size	4	2
Header size	0x14	8
Payload size in the initial handshake packet	Up to 0x3F	HTTP(S): Up to 0x1F, UDP: 0x10

The key for the encoding is included in the header. Every integer value in the header is in big endian. Randomly-sized data will be appended as the payload to the initial handshake packet in both cases.

The immediate values used by the encoding algorithms are different per variant (probably per ShadowPad builder version). Analysis was performed on three ShadowPad variants, which TAU was able to collect in August 2021, as displayed in Table 2. The SHA256 hash values are included in the Indicators of Compromise section below.

Table 2: Analyzed ShadowPad variants

Variant name	**C2 protocol**	**Config size**	**Attribution**	**Source**
Variant1 (aka ScatterBee)	TCP/UDP	0x896	APT41	Positive Technologies

Variant2	HTTP(S)	0x85C	Tonto Team	ESET
Variant3	HTTP(S)	0x85C	unknown	Positive Technologies

TCP Protocol

Analysis was performed to fully detail the C2 protocol. The TCP protocol header format is displayed as follows.

struct struc_common_header

{

__int32 session_key;

__int32 plugin_and_cmd_id; // plugin_id (0x68) << 16 + cmd_id (0x51)

__int32 module_code; // 0

__int32 payload_size_compressed;

__int32 payload_size_original;

};

The header format has been the same since first analyzed in 2015. The session_key is randomly generated and then used for encoding both the header and payload. The plugin_id and cmd_id values included in the plugin_and_cmd_id field have been updated by variants, some of which are covered in this paper. The values in the initial packet created by Variant1 should be 0x68 (Online plugin) and 0x51 (check-in). The module_code of the initial packet generated by the sender is always 0 (zero).

If any payload data exists, it will be compressed with the QuickLZ algorithm. QuickLZ is an older, publicly available compression routine that is not commonly seen. The client generates randomly-sized null bytes (up to 0x3F bytes) for the initial packet payload.

The Variant1's encoding algorithm for the TCP packet in Python is displayed in Figure 1. Based on the protocol analysis results of Variant2 and Variant3, variants of this malware are expected to contain unique immediate values instead of 0x22F4B1BA for the TCP packet encoding.

```
for s in src:
    key = (key - 0x22F4B1BA) & 0xffffffff
    d = (s ^ (key + (key >> 8) + (key >> 16) + (key >> 24))) & 0xff
    _dst.append(d)
```

Figure 1: TCP packet encoding by Variant1

After the initial handshake, Variant1 executes the commands of the plugins specified by the C2 server. For more details, review the Dr.WEB white paper explaining the individual command IDs and payload formats. The variant analyzed in the paper is older than Variant1 but the formats should be similar.

HTTP(S) and UDP Protocols

The header format for the HTTP(S) and UDP protocols is listed below. In HTTP(S), the data is sent through the POST method.

struct struc_proto_header

{

__int16 session_key;

__int16 type; // 0 in HTTP, req=0x1001/res=(0x2002|0x5005) in UDP

__int16 session_src_id; // random 2 bytes, generated by both client/server

__int16 session_dst_id; // req=0, res=client's session_src_id

};

The session_key has the same role as the TCP session_key though the key size is different. The second field type is always 0 (zero) in the HTTP initial packet while the UDP client and server send 0x1001/0x2002/0x5005. The session_src_id field is randomly generated by both client/server. The value sent by the client will be set in the session_dst_id field on the server side.

The initial packet payload data are randomly generated based on QueryPerformanceCounter and other APIs. The HTTP payload size is also random with a length of up to 31 (0x1F) bytes while the UDP one is fixed at 16 (0x10) bytes.

Each of the three Variant encoding algorithms in Python is shown below. The immediate values in the code are different, but the algorithm itself is identical.

```
for s in src:
    tmp1 = (0xCCDD0000 * key) & 0xffffffff
    tmp2 = (0x5A33323 * (key >> 0x10)) & 0xffffffff
    key = (((tmp1 - tmp2) & 0xffffffff) - 0x52B704E3) & 0xffffffff
    d = s ^ (key & 0xff)
    _dst.append(d)
```

Figure 2: UDP packet encoding by Variant1

```
for s in src:
    tmp1 = (0xAD5E0000 * key) & 0xffffffff
    tmp2 = (0x1C1A52A2 * (key >> 0x10)) & 0xffffffff
    key = (((tmp1 - tmp2) & 0xffffffff) - 0x43B69C62) & 0xffffffff
    d = s ^ (key & 0xff)
    _dst.append(d)
```

Figure 3: HTTP(S) packet encoding by Variant2

```
for s in src:
    tmp1 = (0x8D7B0000 * key) & 0xffffffff
    tmp2 = (0x633D7285 * (key >> 0x10)) & 0xffffffff
    key = (((tmp1 - tmp2) & 0xffffffff) - 0x7950BEA0) & 0xffffffff
    d = s ^ (key & 0xff)
    _dst.append(d)
```

Figure 4: HTTP(S) packet encoding by Variant3

After the initial handshake, the payload will contain the same data structure as the TCP packet (struc_common_header and its QuickLZ-compressed payload) explained in the previous section while the type field value in the struc_proto_header will be incremented.

Scanner Implementation

TAU decided on the following target protocols/ports based on the configurations extracted from the recent ShadowPad samples. As explained earlier, the scanner per variant had to be implemented due to the difference in immediate values used in the encoding.

Table 3: Target protocols/ports

Scanning start period	**Target protocol/port/variant**
September 2021	HTTP/443 (Variant2 & Variant3)
October 2021	TCP/443 & UDP/53 (Variant1)
June 2022	UDP/443 (Variant1), HTTP/80 (Variant3)

The following flow chart shows how the ShadowPad C2 servers are detected by the scanners.

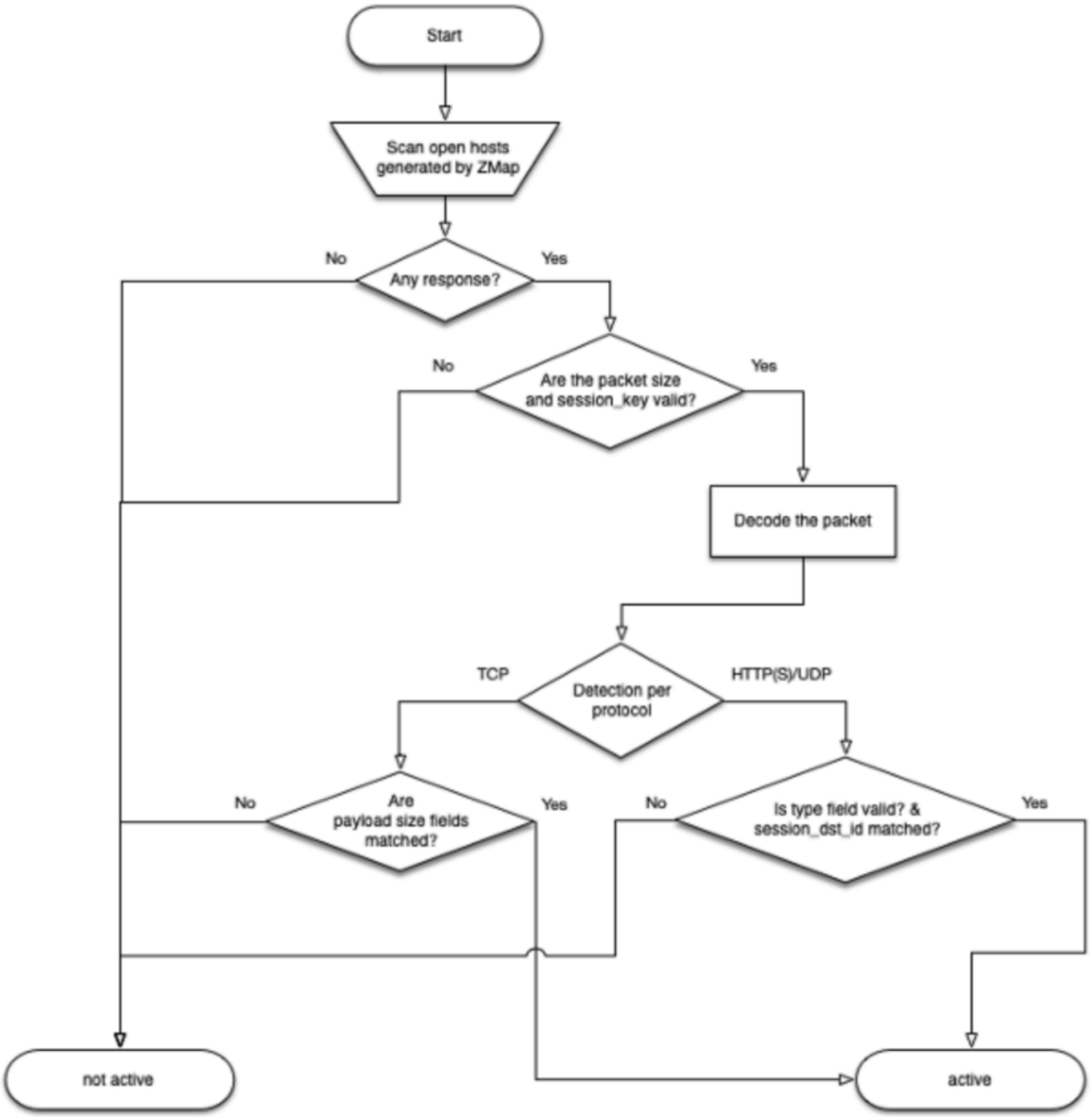

Figure 5: ShadowPad C2 detection flow

Similar to our Winnti 4.0 C2 scanning research, first the list of hosts open at targeted ports are created by ZMap. Then the scanner sends the ShadowPad-formatted packets to all IP addresses on the list. Next, the scanner checks that the response packet size is at least more than the header size and the session_key is different from the sending one to exclude honeypots. If the size and key look to be valid, the scanner decodes the response packet. In TCP protocol, the scanner validates the payload size fields (payload_size_compressed and payload_size_original). In HTTP(S) and UDP protocols, the code verifies if the type field value is correct and the response's session_dst_id is matched with the session_src_id created by the scanner.

The following output log shows that eight Variant1 TCP servers were discovered by scanning the list of TCP/443 open hosts generated by ZMap. The command_id 0x53 from the C2s is a request to send system information of the infected host.

2022/06/xx xx:00:02,log file opened: scan_results/sp_scan_auto_202206xx_xxxxxx.csv

2022/06/xx xx:00:05,malware options: family = ShadowPad; targeted protocol = tcp (version = Variant1)

2022/06/xx xx:00:09,ShadowPad specific options: version = Variant1; key size = 4; key endian = big; header size = 0x14; Online plugin ID = 0x68; CMD ID = 0x51; module code = 0x0

2022/06/xx xx:00:16,51576779 open hosts read from corpus/2022-xx-xx_zmap22000ppsVPN_tcp_443.saddr

2022/06/xx xx:43:46,45.137.10.3,active,compressed payload size matched (plugin_id=0x68, command_id=0x53, payload=None)

2022/06/xx xx:40:28,45.32.248.92,active,compressed payload size matched (plugin_id=0x68, command_id=0x53, payload=None)

..[SKIPPED]..

2022/06/xx xx:01:05,43.129.188.223,active,compressed payload size-matched (plugin_id=0x68, command_id=0x53, payload=None)

2022/06/xx xx:48:35,51576779 scanned in 1 day, 17:48:32.497550

2022/06/xx xx:48:35,8 suspicious/active servers found (DB new=4 update=4)

In order to detect the Variant2/Variant3 C2 servers TAU just uses the HTTP protocol scanner, not the HTTPS one, because the ShadowPad C2s can accept multiple protocol requests at a single port. TAU noticed the unique feature by extracting the C2 server configurations from the sample (SHA256: d011130defd8b988ab78043b30a9f7e0cada5751064b3975a19f4de92d2c0025).

[*] config size = 0x85c

..

[+] C2 Entry 0 (offset 0xbc): ‘HTTPS://wwa1we.wbew.amazon-corp.wikaba.com:443’

[+] C2 Entry 1 (offset 0xed): ‘HTTP://wwa1we.wbew.amazon-corp.wikaba.com:443’

..

The hostnames and ports in the entries matched exactly but the protocols were different. In fact, TAU could verify that another active ShadowPad C2 can accept both protocols at the same port.

$ ./c2fs.py -d -l corpus/query.txt -p 443 -f sp http Variant2

..

[*] malware options: family = ShadowPad; targeted protocol = http (version = Variant2)

[*] ShadowPad specific options: version = Variant2; key size = 2; key endian = big; header size = 0x8; header type = 0x0; client session ID = 53978

[D] POST: http://137.220.185.203:443/ (proxy={}, stream=True, timeout=30)

[+] 137.220.185.203,active,client session ID matched (type=0x0)

..

$ ./c2fs.py -d -l corpus/query.txt -p 443 -f sp https Variant2

..

[*] malware options: family = ShadowPad; targeted protocol = https (version = Variant2)

[*] ShadowPad specific options: version = Variant2; key size = 2; key endian = big; header size = 0x8; header type = 0x0; client session ID = 52256

[D] POST: https://137.220.185.203:443/ (proxy={}, stream=True, timeout=30)

[+] 137.220.185.203,active,client session ID matched (type=0x0)

..

The same behavior may be seen in other protocol combinations such as TCP/SSL and UDP/DNS. However, it's impossible to test because TAU has not obtained any samples of the variants with the multiple C2 protocol plugins yet.

Result

Between September 2021 to September 2022, TAU identified 83 ShadowPad C2 servers (75 unique IPs) on the Internet. The percentage of each variant is shown in Figure 6. During the tracking period, we witnessed that Variant1 had become more active.

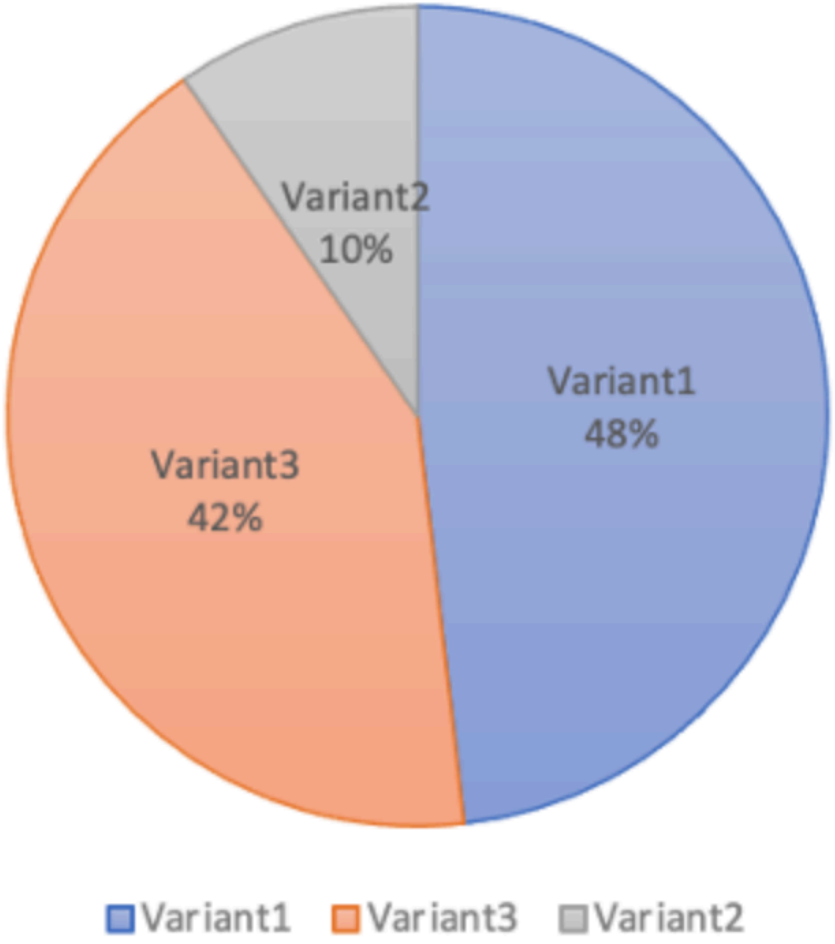

Figure 6: ShadowPad population by variant

The change in the number of active ShadowPad C2s is shown in Figure 7.

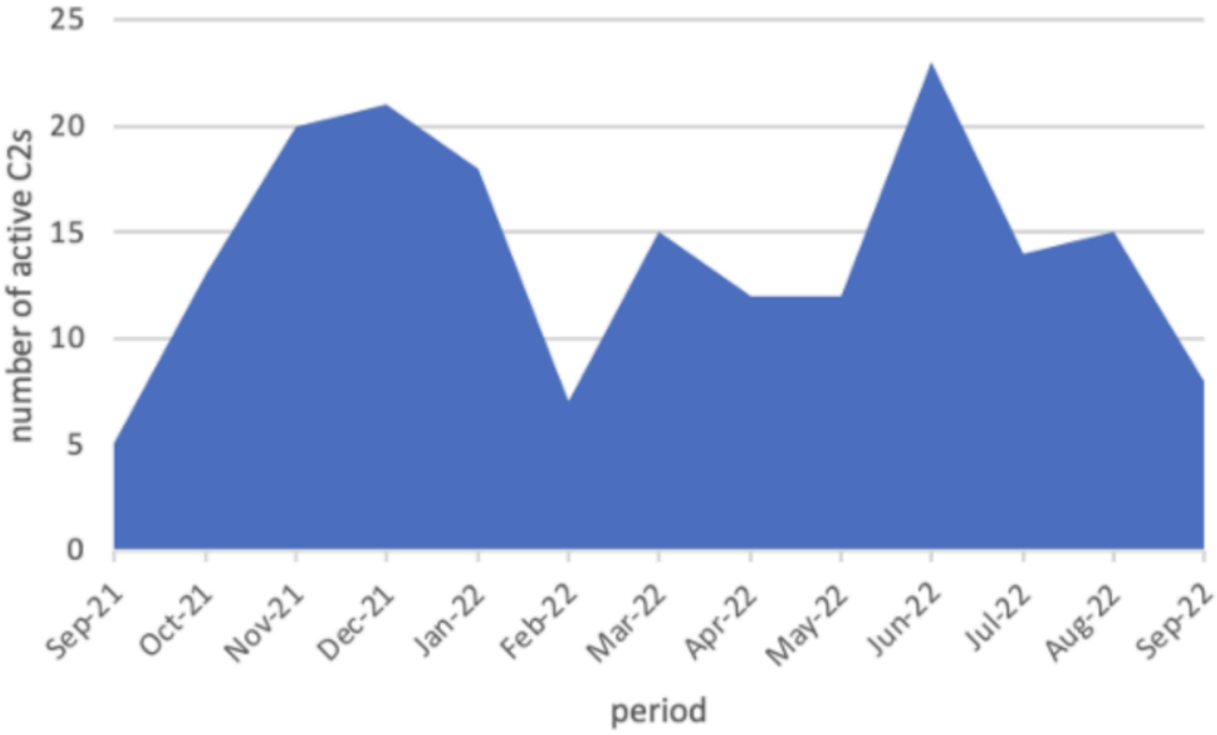

Figure 7: Change in the number of active ShadowPad C2s

Compared with 2021, the active C2s in 2022 has been on a declining trend, though the sharp drop in February 2022 was due to the system issue. The scanner may have missed a new variant lately as ShadowPad changes the immediate values used in the packet encoding per variant. TAU will continuously improve the scanner as TAU obtains new variant samples.

Malware Samples Sharing C2 IPs

TAU identified three samples communicating with the ShadowPad C2 IP addresses on VirusTotal. The sample information is listed in Table 4.

Table 4: Samples communicating with the ShadowPad C2 IPs

Sample Malware family	**C2 IP address**	**C2 Protocol/Port used by sample**	**Sample submission date**	**C2 first-seen date by scanner**	**C2 last-seen date by scanner**
Spyder	156.240.104.149	TLS/443	2021/10/26	2021/10/16	2021/10/16
ReverseWindow	43.129.188.223	TCP/10333	2022/02/27	2021/10/17	2022/10/04
ShadowPad	213.59.118.124	UDP/443	2022/03/20	2022/03/06	2022/09/27

Spyder and ReverseWindow are APT malware utilized by PRC-linked cyber espionage threat actors (respectively APT41 and LuoYu). All C2s were discovered by the TCP/443 Variant1 scanner, but the samples communicated

with a different protocol or port. Except the Spyder sample case, the C2s had accepted multiple protocols/ports at that time. The scanning system caught the C2s prior to the sample submissions in all cases.

Spyder Code Similarity with Winnti 4.0

Incidentally, it should be noted that the above-referenced Spyder sample contains the code handling the same C2 command data structure as Winnti 4.0 Worker which TAU reported three years ago in 2019.

Figure 8: Code handling C2 commands

The command IDs used by the malware families are shown in Table 5. The commands are decided based on a combination of two numbers. Dr.WEB defined the numbers as tag and id in the Spyder report while TAU defined them as cmd_ID and dispatch_ID in the Winnti 4.0 Worker analysis. Both have almost the same C2 command functions.

Table 5: Spyder and Winnti 4.0 command IDs

Command	Spyder		Winnti 4.0 Worker	
	tag	id	cmd_ID	dispatch_ID
Verify the client	1	1	1	1
Send victim information	5	3	5	1
Send plugins information	6	1	6	9 or 13
Save plugin parameters	6	2	6	2
Save plugin data	6	3	6	3
Load and run plugin entrypoint and export function #1	6	4	6	6
Run plugin export function #4 and unload the plugin	6	5	6	7
Heartbeat	6	6	6	8

Run plugin export function #2	6	7	6	10
Run plugin export function #3	6	8	6	11
Send current connection information	7	2	– (no command)	–
Run function pointer of the 2nd parameter obtained by running export function #1	11	–	11	–

On the other hand, the total code similarity between them is just 37% when analyzed with the BinDiff utility. Other data structures like configuration block and C2 protocol header are much different. Based on the comparison displayed in Table 5, TAU hypothesizes that Spyder is a lightweight version of Winnti 4.0 Worker.

Table 6: Comparison of Spyder and Winnti 4.0 Worker

	Spyder	**Winnti 4.0**
Payload encoding / encryption	**single-byte XOR**	**AES in CTR mode (key given as a cmdline argument)**
C2 Protocol	**TLS**	**TCP/TLS/HTTP(S)/UDP**
Server-mode support	**No**	**Yes**
3rd-party library	**uthash, Mbed TLS**	**uthash**
Reported year	**2020**	**2019**

Endpoint Detection

Last year the discovery of the use of a discovered C2 IP (107.155.50.198) triggered an incident response. The advanced and sophisticated attack had bypassed many methods of detection but was ultimately alerted upon simply because of the pre-identified C2 IP.

Figure 9: Alert based on the ShadowPad C2

Conclusion

By emulating the ShadowPad C2 protocols then scanning the C2 servers on the Internet, TAU has discovered over 80 C2 servers. The IOCs has been published on the GitHub page with discovered date ranges which are more helpful than just IP address information since the C2s are typically found on hosted servers. Approximately 10 C2s have always been active. TAU sees little possibility of false positives because the C2 protocol formats and encoding algorithms are fairly unique.

Scanning APT malware C2s on the Internet is sometimes like finding a needle in a haystack. However, once the C2 scanning works, it can become a game changer as one of the most proactive threat detection approaches.

Acknowledgment

TAU appreciates Leon Chang's expertise and advice regarding ShadowPad. Chang shared his knowledge to gain a more complete, bigger picture of the variants.

Indicators of Compromise (IOC)

Indicator	Type	Context
03b7b511716c074e9f6ef37318638337fd7449897be999505d4a3219572829b4	SHA256	ShadowPad Variant1
aef610b66b9efd1fa916a38f8ffea8b988c20c5deebf4db83b6be63f7ada2cc0	SHA256	ShadowPad Variant2
d011130defd8b988ab78043b30a9f7e0cada5751064b3975a19f4de92d2c0025	SHA256	ShadowPad Variant3
1ded9878f8680e1d91354cbb5ad8a6960efd6ddca2da157eb4c1ef0f0430fd5f	SHA256	Spyder communicating with the

		ShadowPad C2 (156.240.104.149)
536def339fefa0c259cf34f809393322cdece06fc4f2b37f06136375b073dff3	SHA256	ReverseWindow communicating with the ShadowPad C2 (43.129.188.223)
9447b75af497e5a7f99f1ded1c1d87c53b5b59fce224a325932ad55eef9e0e4a	SHA256	ShadowPad Variant1 communicating with the ShadowPad C2 (213.59.118.124)
