KASPERSKY lab

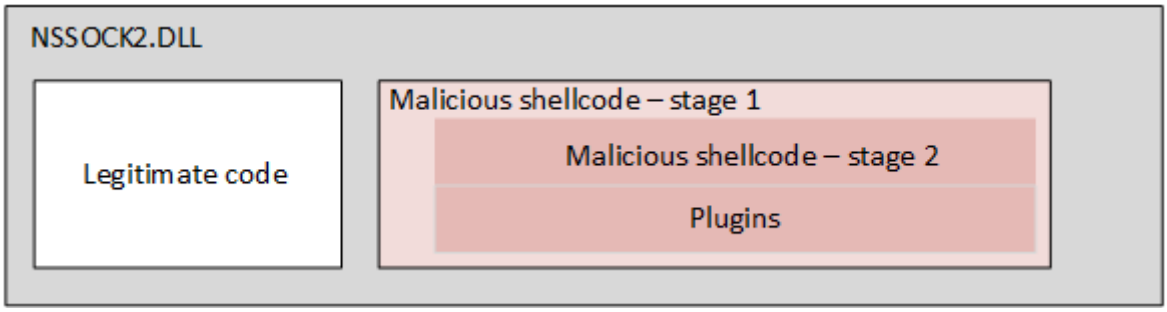
ShadowPad: popular server management software hit in supply chain attack

Part 2: Technical Details

ShadowPad is a modular cyber-attack platform that attackers deploy in victim networks to gain flexible remote control capabilities. The platform is designed to run in two stages. The first stage is a shellcode that was embedded in a legitimate nssock2.dll used by Xshell, Xmanager and other software packages produced by NetSarang. This stage is responsible for connecting to “validation” command and control (C&C) servers and getting configuration information including the location of the real C&C server, which may be unique per victim. The second stage acts as an orchestrator for five main modules responsible for C&C communication, working with the DNS protocol, loading and injecting additional plugins into the memory of other processes.
All actual payloads are received from the real C&C as plugins and can perform different types of data exfiltration.

NSSOCK2.DLL - the compromised library

```
SHA256          462a02a8094e833fd456baf0a6d4e18bb7dab1a9f74d5f163a8334921a4ffde8
MD5             97363d50a279492fda14cbab53429e75
Compiled        2017.07.13 01:23:01 (GMT), 11.0
Type            I386 Windows GUI DLL
Size            180432
Internal name nssock2.dll
```



The main loader is built into the original "nssock2.dll", which is digitally signed. The malicious code is triggered from one of the object autoinitializations that are automatically called by the C runtime code. It decrypts a binary blob with a function similar to “rand” and directly starts its execution.

The blob is a self-loading executable converted into shellcode. It starts with a loader that processes a proprietary PE-like formatted blob, loads the code and data section by section, resolves imported API functions, relocates the code and then calls the entrypoint as DllMain.

Each self-loading shellcode contains a timestamp field that appears to be equal to UNIX timestamps.

Shellcode in NSSOCK2.DLL

```
Size            77824
Type            shellcode, binary reconstructed from a proprietary format
```

```
Timestamp    2017.05.26 07:00:23 (GMT)
```

The binary object is produced from a compiled Windows DLL file. The entrypoint of the blob starts with a standard Microsoft Visual Studio DllMain code stub. All strings are encrypted with a custom randomisation function and each string starts with a 2-byte decryption key. It maintains a configuration block in the Windows registry using one of the following locations:

HKCU\SOFTWARE*%d*
or
HKLM\SOFTWARE*%d*,

where *%d* is a signed integer produced from the system drive's serial number *xor-ed* with 0xD592FC92. The block is stored in a value named "*Data*" and is 552 bytes long. It contains a unique user id (generated GUID), 8 byte decryption key for the second stage, first execution time, execution counter. It generates a hostname for accessing its C&C server using a DGA (domain generation algorithm) based on the current month and year in the *.com* top level domain. The request to the C&C is sent through the DNS extracted from the network adapter settings or to hardcoded DNS servers IPs : 8.8.8.8, 8.8.4.4, 4.2.2.1, 4.2.2.2.

As of August 2017, the following domain name was used: **nylalobghyhirgh.com**
At the time of analysis the domain was registered with the following WHOIS information:

```
Domain Name: NYLALOBGHYHIRGH.COM
Registry Domain ID: 2146218329_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.namesilo.com
Registrar URL: http://www.namesilo.com
Updated Date: 2017-07-24T06:41:22Z
Creation Date: 2017-07-24T06:41:22Z
Registry Expiry Date: 2018-07-24T06:41:22Z
Registrar: NameSilo, LLC
Registrar IANA ID: 1479
Registrar Abuse Contact Email: abuse@namesilo.com
Registrar Abuse Contact Phone: +1.4805240066
Domain Status: clientTransferProhibited
https://icann.org/epp#clientTransferProhibited
Name Server: NS1.QHOSTER.NET
Name Server: NS2.QHOSTER.NET
Name Server: NS3.QHOSTER.NET
Name Server: NS4.QHOSTER.NET
DNSSEC: unsigned
```

DNS requests are sent every 8 hours. The request buffer presented to the C&C server contains the following data:

```
Value               Description
-----------------------------------
00 00           encryption key
```

```
52 4F 4F 44          'DOOR', magic value
6 bytes          GUID
2 bytes          iteration counter
1 byte           year's least significant byte + 0x30
1 byte           month
1 byte           day
*                hostname
*                domain name
*                user name
```

The request buffer is encrypted with custom XOR-based encryption algorithm. Then, the encrypted buffer is converted to a readable string of latin characters by adding each half of the byte to ‘a’ and ‘j’ characters correspondingly.
The first character is encoded by adding the 'a' character to the number of non-dot characters in the DGA domain name. This string is split in a series of subdomains of 50-63 bytes long split by dots and then prepended to the DGA-generated hostname name.

The resulting request packet querying a **.....*.%DGA-domain%*.com is sent to all the DNS servers available and Google DNS servers.

The DNS packet wrapping the request buffer starts with the following fields:

```
Value              Description
-------------------------------------
2 bytes          Random request ID
01 00            Opcode (recursive request)
00 01            Number of queries: 1
00 00            Number of answers: 0
00 00            Number of name server records: 0
00 00            Number of authoritative response records: 0
*                Encoded data represented as a hostname
00 10            Query type: TXT
00 01            Query class: IN
```

The module waits for a response from any DNS server until timed out or some data is received. The DNS packet is checked to conform to the following format:

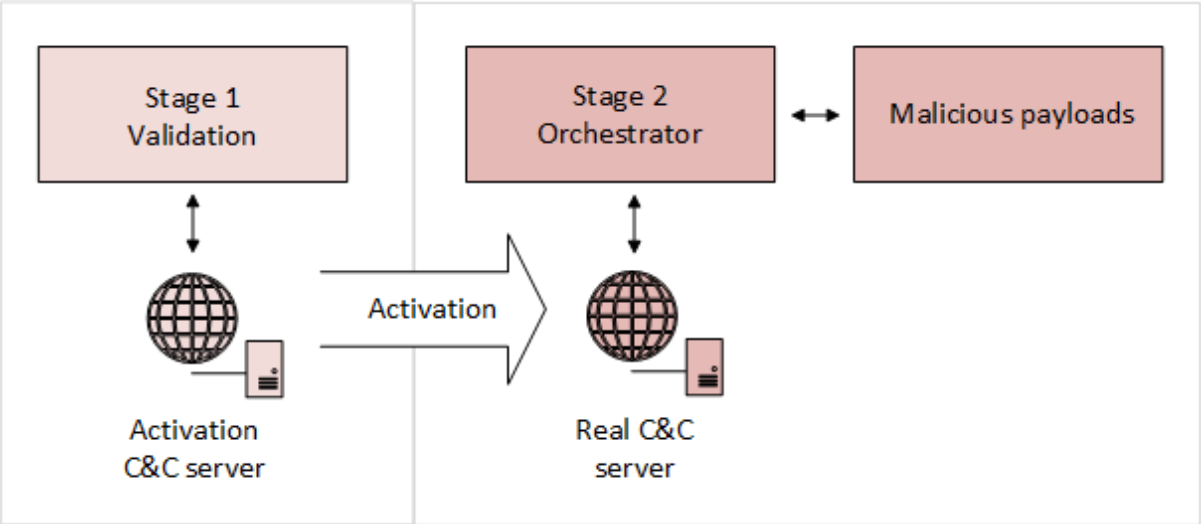
```
Value               Description
-------------------------------------
?? ??            ID
xx x0            No error
xx xx            Number of queries
xx xx            Number of answers
xx xx            Number of name server records
00 00            Number of authoritative response records
```

```
--query--
*               Encoded data in the query record, first char + 'a'
is number of stray bytes at the end, all the rest encoded as two
latin characters starting from 'a', 'j'
00 10           Query type: TXT
00 01           Query class: IN
--response--
C0 0C           Name: backwards link to the query record
00 10           Query type: TXT
00 01           Query class: IN
00 01 xx xx     TTL
xx xx           Record length
*               Encoded response from the C2 server. Data format is
the same as for the query part.
```

The response string is decrypted using the first two bytes of the response packet as a key. The data format follows:

```
xx xx           encryption key
-- after decryption --
52 4F 4F 44          'DOOR', magic value
2 bytes         iteration counter
1 byte          status : 1 - ready to decrypt the payload, 2 - stop
operation
4 bytes         part of decryption key
4 bytes         part of decryption key
4 bytes         length of additional data
*               additional data
```

The module copies information received from the C&C server to its configuration storage and updates the corresponding registry key.
Once a proper decryption key is read from the registry or from the DNS response it is used to decipher the second encrypted shellcode (“stage 2”). It is then called directly passing the 'additional data' string received from the C&C as an argument to the shellcode.



Second stage shellcode in “NSSOCK2.DLL”

```
Size          54288
Type          shellcode, binary reconstructed from a proprietary format
Timestamp     2017.05.26 07:00:13 (GMT)
```

The shellcode was produced from a Windows DLL file. The entrypoint of the blob starts with standard MSVC DllMain code. The format of the blob is the same as in the "Shellcode in NSSOCK2.DLL".
The DllMain function differs from the standard C/C++ implementation. Besides standard “fdwReason” parameter values it also processes custom ones: 100, 101, 102, 103, 104.

"Reason" codes 102-104 are used to implement a custom plugin API.

Code 100 : plugin initialization
Code 101 : plugin deinitialization
Code 102 : return the plugin's numeric identifier 100 in the lpReserved parameter
Code 103 : allocate a string for the plugin's name i.e. "Root" and return the value in the lpReserved parameter
Code 104 : return a pointer to plugin's function table in a DWORD pointed by the “lpReserved” parameter

During DLL initialization the module allocates memory for internal structures and sets up its function table. Then, it uses the pointer to its own image as a plugin and initializes the plugin infrastructure.
The plugin is started by calling sequentially its entrypoint (DllMain) with “Reason” parameters 100, 102, 104 and copying the data returned in a structure describing the plugin. Once the plugin returns no error during initialization it is added to the plugin list.

Depending on the mode of operation, it can then proceed with the plugin orchestrator, either in a separate thread or inline - that is specified by the parameter provided by the C&C.

Module / plugin orchestration

The image contains five encrypted blobs structured in the same way as the second stage blob. They are decrypted with a XOR-based algorithm, decompressed with QuickLZ and loaded in memory. Then they are initialized and added to the list of plugins in the same way as the "Root" plugin.
Each plugin has a name and a numeric identifier (ID):

```
ID   Name
-------------
100  Root (the second stage shellcode itself)
101  Plugins
102  Config
103  Install
104  Online
203  DNS
```

Then the module searches for the plugin with ID 103 (“Install”) and calls its second function. The process is terminated if the plugin is not available. The module remains in memory as the "Root" plugin and provides various facilities for the other plugins via the exported function table.

Modules

“Install” module

```
Size         7877
Type         shellcode, binary reconstructed from a proprietary format
Timestamp    2017.05.26 06:59:46 (GMT)
```

Starts by adjusting process privileges, then invokes the “Config” plugin's function "LoadConfig"
If there are no additional parameter from the C&C it continues to the main thread, otherwise it injects into a newly created process and continues from there.

Creates mutex "**Global*%16-48 random latin characters%***"

The module continues by invoking the modules “Plugins” and “Online”.

“Plugins” module

```
Size         7119
Type         shellcode, binary reconstructed from a proprietary format
Timestamp    2017.05.26 06:59:07 (GMT)
```

This plugin provides API for other modules and does not initiate any actions without external intervention.
Starts a registry monitoring thread waiting for changes in its registry key and loads any new plugins available in the registry Virtual File System (VFS).
The registry location is :

HKLM\HKCU\SOFTWARE\Microsoft*%5-12 random characters%*

Every value found in the key is decrypted and checked if it is a valid plugin and then loaded and initialized using the API from the "Root" module.

Also, the plugin provides an API for reading, writing and deleting arbitrary registry values. This also allows for the writing of new plugin images to the registry VFS by command from the C&C server.

"Config" module

```
Size          6574
Type          shellcode, binary reconstructed from a proprietary format
Timestamp     2017.05.26 06:59:16 (GMT)
```

This module maintains a configuration block of data of a fixed size of 2136 bytes. The block consists of a fixed size header and a string pool populated sequentially and referenced from the fixed header. When invoked for the first time during the current session it initializes with a default configuration. The default C&C server URL may be overwritten with the one provided from the the packet used to activate the second stage shellcode, if present.

The configuration string pool starts from offset 0x58 and holds several string parameters. Each string is encrypted with a random 2-byte key using a proprietary algorithm based on XOR and an in-house rand() function. The encryption algorithm is the same for all string constants used in all of the components of the malware.

```
Offset  Size   Value
--------------------------
000        2   Offset of the string constant "HD"
002        2   Offset of the string constant "HD"
010        2   Offset of the executable path used for injection
018        2   Offset of the C&C server URL
040        4   IP address "8.8.8.8"
044        4   IP address "8.8.4.4"
048        4   IP address "4.2.2.1"
04C        4   IP address "4.2.2.2"
050        4   Constant 0x708 - sleep interval, equal to 1800
seconds or 30 minutes
058        *   String pool:
```

```
"HD"  String constant
"HD"  String constant
*     C&C server URL, default value: "dns://www.notped.com", or the
one provided by the server to the "Shellcode in NSSOCK2.DLL" code.
"%windir%\system32\svchost.exe" Path to the executable used as a
host for injection
```

The configuration block is prepended with a service header, compressed and encrypted using a function provided by the "Root" plugin and then written to a file. The resulting size of the file is 2156 bytes.

Service header format:

```
Offset  Size  Value
--------------------------------
000   4     00 00 00 00
004   4     12 34 56 78
008   4     00 00 00 00
00C   4     00 00 08 58 // Size of the configuration block in bytes
010   4     * // Unused
```

The exact location of the configuration file depends on the system volume's serial number and is generated according to the following format:

%ALLUSERSPROFILE%\%random 3-8 latin characters%\%random 3-8 latin characters%\%random 3-8 latin characters%\%random 3-8 latin characters%

The file is always overwritten every time the plugin is initialized.

“Online” module

```
Size         15803
Type         shellcode, binary reconstructed from a proprietary format
Timestamp    2017.05.26 06:59:21 (GMT)
```

Handles overall communication with the C&C server and dispatches the commands to other plugins.

Maintains a registry key : **HKLM\HKCU\SOFTWARE*%random 3-8 latin characters%*** containing a 24-byte record of system time and number of tries.

Processes the list of C&C URLs from the configuration block (up to 16 URLs). Depending on the protocol specified in the URL it selects one of the plugins for handling communication with the C&C server.

```
Protocol  Plugin ID
------------------
```

```
TCP         200
HTTP        201
HTTPS       204
UDP         202
DNS         203
SSL         205
URL         built in, DGA-based HTTP client
```

It maintains a connection using one of the plugins, starting the connection with an initial packet and getting and executing commands and sending result packets back. The data received back is either a shutdown message or a packet of data containing a plugin ID and additional data for the command.

In case of the "URL" protocol, it uses a built-in HTTP client to resolve the actual C&C server URL from an intermediary C&C server. It uses its own DGA based on the day of the month, range (1-10, 11-20, >20) to generate the name of the intermediary C&C server.
The actual name of the server is based on a mask specified in the configuration data, i.e. prefix*%DGA-generated part%*suffix, and the location of the suffix is marked with a '@' char.

Depending on the URL scheme specified in the mask it selects FTP, HTTP or HTTPS protocol to send the request to the intermediary server and either sends a "GET" request or fetches a file from FTP.
Once it has received a response the module looks for a string framed with '$' characters. The string is then decoded into a binary buffer by subtracting 'a' characters and concatenating each pair into one byte. Then the buffer is decrypted using an algorithm that is used for string encryption in the rest of the code. The resulting decrypted buffer is expected to be the actual URL of the C&C to use then.

The module may also provide basic information about the system when requested by the C&C server:

- current date and time
- memory status
- CPU frequency
- amount of free disk space
- video mode
- system locale
- PID of the malicious process
- OS version
- domain name
- user name

“DNS” module

```
Size          10982
Type          shellcode, binary reconstructed from a proprietary format
Timestamp     2017.05.26 06:58:11 (GMT)
```

Handles all C&C communication based on the DNS protocol.

Sends and receives DNS TXT records messages in the same way as the "Shellcode in NSSOCK2". However, the encoded payload is decrypted using a different in-house algorithm and the format of the response buffer is different:

```
Offset  Size  Value
------------------
000   2     Encryption key
-- after decryption --
002   2     Packet type(0,1,3)
004   2     packet id1 (of the server's response)
006   2     packet id2 (of the packet server is responding to, ACK)
```

Initial packet, type 0:

```
Offset  Size  Value
------------------
000   2     Encryption key
002   2     00 00
004   2     packet id 1
006   2     packet id 2
008   16    GUID
```

Packet type 1 - data:

```
Offset  Size  Value
------------------
000   2     Encryption key
002   2     00 01
004   2     packet id 1
006   2     packet id 2
008   *     payload
```

Packet type 3 - shutdown message:

```
Offset  Size  Value
------------------
000   2     Encryption key
002   2     00 03
004   2     packet id 1
006   2     packet id 2
```