From Bear to Vault: Designing a New Protocol to Extend the APT Communications Toolset

Nick Hendee
Department of Computing Security
Rochester Institute of Technology
Rochester, United States of America
kp.ai@protonmail.com

Christopher Partridge
Department of Computing Security
Rochester Institute of Technology
Rochester, United States of America
chris@partridge.tech

***Abstract*—With the growing complexity and prevalence of state-sponsored Advanced Persistent Threats (APTs), exfiltration of bulk sensitive data is becoming increasingly commonplace. Furthermore, recent APTs and APT toolkits uncovered have been utilizing creative methods to communicate with Command and Control (C2) infrastructure as well as exfiltrate data, often using multiple hard-to-detect methodologies such as trusted third-party services, SMTP, or DNS. This paper proposes a scalable protocol for managing multiple secure and reliable covert channels over physical or digital means. Throughout we will show the desire for, practical applications of, and future goals for this protocol.**

***Index Terms*—covert channel, protocols, advanced persistent threats, encipherment, security**

I. Introduction

In 2013, FireEye broke ground by releasing a detailed report on a likely state-sponsored Advanced Persistent Threat (APT) group originating in China responsible for over 140 independent breaches and exfiltrating hundreds of terabytes of data [1]. This was not the first activity of APT groups, as APT activity stretches as back as far as 2006 [2], but it would be far from the last. Five years later, the movements or emergence of APT groups is commonplace, and publicly-known active threats include APT38 [3], Reaper [4], Fancy Bear [5], MenuPass Group [6], OceanLotus Group [7], APT33 [8], and Group 123 [9], among others.

APTs have a unique need for covert channels, as they must establish and maintain long-term access to compromised systems without arousing suspicion, often opting to deploy beacons that utilize Command and Control (C2) infrastructure [10]. This is further complicated by certain APTs needing to exfiltrate data in bulk [10] - a noted shortcoming of many traditional covert channels, whose throughput is often measured in bits per second [11] even when the immense speed of Internet protocols is accounted for [12]. Thus, APTs needing to exfiltrate large quantities of data use many different methods, such as compromising trusted websites [13], abusing trusted third-party cloud services, or mimicking trusted channels such as TLS [9].

This paper evaluates previous work in covert channels, and reviews how contemporary adversaries have been using covert channels in their ongoing campaigns. Further, we elucidate a new protocol for use in advanced threat simulation, as well as extends on known capabilities to increase speed, reliability, and flexibility of both data exfiltration and C2 tasks. Finally, we present our implementation's preliminary results, along with our plans and considerations for further works.

Rochester Institute of Technology

II. Literature Review

A. Traditional Channels

The goal of covert channels was introduced by Lampson in 1973 [14] and extended to networks by Girling in 1987 [15]. Traditionally these channels fall into two categories: storage channels, in which the sender embeds data for the receiver to extract; and timing channels, which signal information by modulating a resource over time [16]. The accepted communication model for covert channels, per Simmons [17] is a prisoner problem: where two prisoners attempting to escape must exchange messages while monitored by the warden, who would put both prisoners in solitary confinement (thus eliminating the chance of escape) should they became suspicious.

Many storage channels exist in the myriad of network protocols such as [15] [18], and many focus on embedding limited amounts of information in headers or metadata of network protocols. Similarly, timing channels exist in network protocols [19] [20], and focus on delaying network traffic in order to signal data. A number of improvements have been proposed to increase the speed or covertness [21] of these channels, including timing coding [22] or entirely new models [23]. These traditional methods adhere to the prisoner's problem exactly, assuming that the warden is manually inspecting all communications between the two prisoners - in this case, networked machines.

B. APT Behaviors

In practice this is often not the case, as the volume of traffic between networks is simply too high for manual inspection, so the 'warden' of secure networks is often an Intrusion Detection System (IDS) or Intrusion Prevention System (IPS), often deployed at the edge of network trust zones [24]. So long as the IDS/IPS does not flag traffic as suspicious (invoking manual review), data exfiltration or C2 communications remain undiscovered.

978-1-7281-1360-9/18/$31.00 ©2018 IEEE
DOI 10.1109/CSCI.2018.00028

Because of this, covert channels selected by APTs tend to be built on standard network protocols that are common to their target's environment and mimic normal user activity, sacrificing covertness for simpler IDS/IPS evasion and speed. A large number of APTs leverage DNS [7] [25], HTTP [5] [6], HTTPS [9] [26], and SMTP [5] - all of which are normally found in business environments. Many will also utilize covert channels at the web application level, for example Reaper [4] has been observed leveraging AOL IM, pCloud, and Dropbox.

Additionally, some APTs have become increasingly efficient, for example APT38 opting to use modular malware [26] and the CIA opting to write communications programs such as Aeris that integrate seamlessly into their C2 software [27]. Some have even written tailored communications solutions to route data through target networks over designated pivots, such as Fancy Bear's [28] XTunnel [29], abusing the failure to inspect traffic on a target's local network. Adversary simulation tools have grown similarly, with Cobalt Strike implementing hybrid communications methods that allow it to communicate over HTTP and DNS in parallel, increasing efficiency and speed [30]. Cobalt Strike's methodology is proven as well, with real-world APTs such as OceanLotus Group opting to utilize parts of its software in the wild [31].

III. Proposed Architecture

A. System Overview

To better solve some of the exfiltration problems that APTs have, we propose a protocol titled "the Asynchronous Remote Communications Tunneling Protocol" or ARC-TP. This protocol is designed to enhance covert channels for C2 and data exfiltration that contemporary APTs are known to use, enabling high data rates, greater flexibility and resilience, and increased efficiency. Additionally, ARC-TP is designed to be part of a modular system, acting as a middleman between application logic and the channels that data is being exfiltrated over. This allows an attacker to write simple channel implementations and use a wide variety of channels, while ensuring that data is exfiltrated in a secure and reliable manner.

ARC-TP is composed of three distinct parts: the wrapper specification, Arc manager, and Arc handler. The Network Architecture section will focus on the wrapper specification, while the System Architecture section will focus on the Arc manager and handler.

B. Network Architecture

There are two types of packets that are used by ARC-TP, Negotiate and Message packets. Negotiate packets are used to create channels, while Message packets contain encrypted data. Both are to be sent over covert channels in the same way: sending data by being queued into a wrapper's "send" queue, be that a file, location in memory, etc. for the wrapper to dequeue and encode into a channel; and receiving data in the inverse way, by being read from the channel it was encoded into, then enqueued into a wrapper's "received" queue. When a wrapper dequeues the message, it should encode the message data into whatever channel it wishes to use, and then send the data over that channel to a receiving wrapper. For example, a wrapper that uses DNS could encode data into a subdomain of a common domain and query a remote wrapper that would read those messages.

Before data can be exhanged, the client and server must first agree on specifications, encryption ciphers, and connection parameters through exchange of negotiation packets. There are several flags available to set in a negotiate packet, as seen in Table I.

TABLE I
NEGOTIATE FLAGS

Flag	**Flag Data**	
ID	***Title***	***Bitmask***
NEGO	Negotiate	0x0800
CONF	Confirm	0x0001
CLONE	Clone	0x0080

The NEGO flag denotes this is a negotiate packet. The CONF flag is set when a server accepts a client's negotiation request, and is returning its own verification data for key establishment and authentication. The CLONE flag is used for setting up additional channels sharing a single stream ID.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|version|       flags           |  encrypt/mac  |    jitter     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| packet count  |               ...Data...                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 1. The format of a Negotiate packet.

As you can see in Fig. 1, we are leaving space in the flags section of the packet to allow for future expansions. Additionally, you can see that we are including a brief "version" field to allow for version handling, an encryption and MAC selection field so the handlers can set what cryptographic protocols are being used, a "jitter" field to denote timeout information, a packet counter for easier ordering and acknowledgment, and the data field.

Once a negotiation has occurred successfully and session data is established, ARC-TP may begin exchanging information through Message packets. The Message packet has more flags available, for more granular control over how Arc handlers interpret data, state changes, etc.

TABLE II
MESSAGE FLAGS

Flag	**Flag Data**	
ID	***Title***	***Bitmask***
SOM	Start-of-Message	0x0100
EOM	End-of-Message	0x0200
EOS	End-of-Stream	0x0400
ACK	Acknowledge	0x0002
WRITE_FILE	Write-to-file	0x0010
SET_JITTER	Set-Jitter	0x0040

The SOM flag denotes that a new message is being started, and while this flag is set the WRITE_FILE flag may also be invoked to tell the Arc handler to write data to file instead of to memory, in the event that a large quantity of data is being sent. The EOM flag denotes that a message is being finalized, and should all components of that message have been transferred from one Arc handler to another, a callback can be executed to inform application logic that a full message has been received. The EOS flag denotes that the session is being terminated, so when communication finishes the Arc wrapper may terminate. The ACK flag denotes that messages are being acknowledged in this message, providing both proof-of-delivery and the ability to automatically resend on failure. Finally, the SET_JITTER flag (similar to Aeris [27]) allows the Arc handler to set a different timeout, allowing for slower transmissions over low-bandwidth channels.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|version|       flags           |           stream id           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| packet count  |     optional headers/data      |  ...Data...   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           message authentication code (if in use)              |
|                             ....                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 2. The format of a Message packet.

Like the Negotiate packet, the Message packet in Fig. 2 includes lengthy version and flag fields, now with a stream ID (established when a session is successfully negotiated), a data section, and an optional MAC section.

C. System Architecture

The Arc handler handles session negotiation, session management, cryptographic services, channel management, and data acknowledgment. It is a fast, two-way communications handler that uses an API for data input and executes asynchronous callbacks for data output to application logic. Additionally, each arc handler may interact with many wrappers simultaneously, and the use of session keys allows multiple arc handlers to be running at the same time, allowing multiple applications to use independent arc handlers on one system.

The Arc handler, seen in Fig. 3, has three main routines: Client Initialization, Send Data, and Receive Data. The Arc handler pulls from and pushes to many wrappers for covert channels in a round-robin manner after initializing a connection over a single channel. While this does not provide channel balancing built-in (ex. utilization of a slow DNS channel vs a fast HTTPS channel), the problem can be mitigated by using multiple wrappers of a higher channel speed. Additionally, the Arc handler is designed to handle cryptographic operations inline with other functionality, encouraging but not forcing the use of encryption so that messages are secured from observers.

The Arc manager is meant to extend the Arc handler and is used for managing negotiation and session creation on data exfiltration or C2 servers, making it easier to globally apply settings changes.

As you can see in Fig. 4, the Arc manager abstracts negotiation overhead away from Arc handlers on remote servers such as C2 or data exfiltration warehouses, and automatically spawns Arc handlers for established sessions. It effectively serves as a new connection listener over a number of wrappers, so there is no overhead in user's application logic.

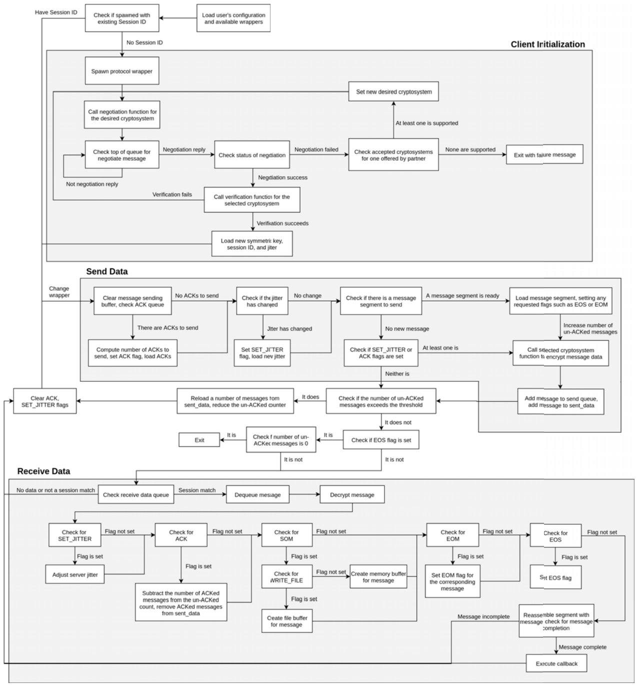
Have Session ID
Check if spawned with existing Session ID
Load user's configuration and available wrappers
No Session ID
Client Initialization
Spawn protocol wrapper
Set new desired cryptosystem
Call negotiation function for the desired cryptosystem
At least one is supported
Check top of queue for negotiate message
Negotiation reply
Check status of negotiation
Negotiation failed
Check accepted cryptosystems for one offered by partner
None are supported
Exit with failure message
Not negotiation reply
Negotiation success
Verification fails
Call verification function for the selected cryptosystem
Verification succeeds
Load new symmetric key, session ID, and jitter
Send Data
Change wrapper
Clear message sending buffer, check ACK queue
No ACKs to send
Check if the jitter has changed
No change
Check if there is a message segment to send
A message segment is ready
Load message segment, setting any requested flags such as EOS or EOM
There are ACKs to send
Jitter has changed
No new message
Increase number of un-ACKed messages
Compute number of ACKs to send, set ACK flag, load ACKs
Set SET_JITTER flag, load new jitter
Check if SET_JITTER or ACK flags are set
At least one is
Call selected cryptosystem function to encrypt message data
Neither is
Clear ACK, SET_JITTER flags
Reload a number of messages from sent_data, reduce the un-ACKed counter
It does
Check if the number of un-ACKed messages exceeds the threshold
Add message to send queue, add message to sent_data
It does not
Exit
It is
Check if number of un-ACKed messages is 0
It is
Check if EOS flag is set
It is not
It is not
Receive Data
No data or not a session match
Check receive data queue
Session match
Dequeue message
Decrypt message
Check for SET_JITTER
Flag not set
Check for ACK
Flag not set
Check for SOM
Flag not set
Check for EOM
Flag not set
Check for EOS
Flag not set
Flag is set
Adjust server jitter
Flag is set
Subtract the number of ACKed messages from the un-ACKed count, remove ACKed messages from sent_data
Flag is set
Check for WRITE_FILE
Flag not set
Create memory buffer for message
Flag is set
Create file buffer for message
Flag is set
Set EOM flag for the corresponding message
Flag is set
Set EOS flag
Message incomplete
Reassemble segment with message check for message completion
Message complete
Execute callback

Fig. 3. The Arc handler state machine.

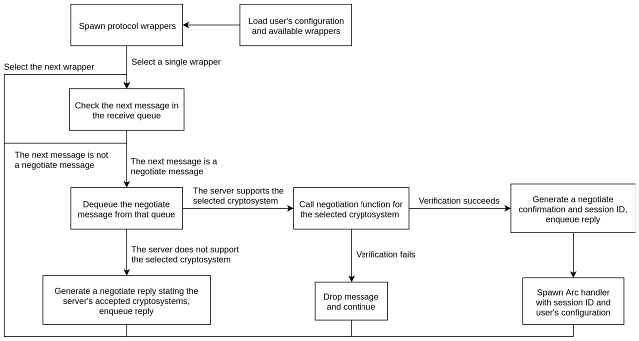
Spawn protocol wrappers
Load user's configuration and available wrappers
Select the next wrapper
Select a single wrapper
Check the next message in the receive queue
The next message is not a negotiate message
The next message is a negotiate message
Dequeue the negotiate message from that queue
The server supports the selected cryptosystem
Call negotiation function for the selected cryptosystem
Verification succeeds
Generate a negotiate confirmation and session ID, enqueue reply
The server does not support the selected cryptosystem
Verification fails
Generate a negotiate reply stating the server's accepted cryptosystems, enqueue reply
Drop message and continue
Spawn Arc handler with session ID and user's configuration

Fig. 4. The Arc manager state machine.

Full-size versions of the above figures are included at `https://github.com/Koschei-Project/arcpub` for your convenience.

IV. Future Work

We have implemented preliminary versions of the ARC-TP protocol in C for Windows, utilizing simple UDP wrappers, without encryption. We have found that the Arc handler overhead is fairly minimal, with under 1 MB of RAM used.

We would like to either expand the Arc protocol to cleanly support tunneling in a way that is similar to XTunnel [29] though this is currently easily achievable through the application layer - simply add a callback to another Arc handler's send data function, and you have created a simple proxy that can easily switch channels. Furthermore, we would like to more clearly define a specification for the application layer to integrate with ARC-TP (similarly to how Aeris integrates with Collide [27]), and provide a robust API specification for implementations of ARC-TP, possibly modelled after Berkeley sockets.

As the protocol solidifies into a well-qualified ”version 1” we will also be doing extended testing of the protocol, both for devising manners of effectively balancing the use of fast and slow wrappers on a single system, as well as testing ARC-TP in hard-to-tackle edge cases, such as extremely high latency channels ex. USB flash disks moving from one location to another, high noise timing channels, and more. We will also be testing to see how many easy-to-detect network signatures this protocol generates, and explore mitigations against signature or rule-based detection.

V. Conclusion

While some parts of ARC-TP have yet to be fully elucidated such as a formalized API for the Arc handler and Arc manager, we believe that this is a step forward in line with how APT groups have been expanding their communications toolkits. Additionally, a flexible and versatile system will greatly assist APTs and advanced threat simulators to adapt to new and changing environments. The ability of this protocol to be easily extended with simple wrappers is a testament to its utility, and future tests will hopefully confirm its design in a wide range of environments and simulations.

References

[1] Mandiant, APT1 - Exposing one of China's Cyber Espionage Units, FireEye. [Online]. Available: https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf.

[2] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains, lockheedmartin.com. [Online]. Available: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[3] N. Fraser, J. O'Leary, V. Cannon, and F. Plan, APT38: Details on New North Korean Regime-Backed Threat Group, FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2018/10/apt38-details-on-new-north-korean-regime-backed-threat-group.html.

[4] FireEye, APT37 (REAPER) The Overlooked North Korean Actor, FireEye. [Online]. Available: https://www2.fireeye.com/rs/848-DID-242/images/rpt_APT37.pdf.

[5] APT28: A Window Into Russia's Cyber Espionage Operations?, FireEye. [Online]. Available: https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-apt28.pdf.

[6] FireEye iSIGHT Intelligence , APT10 (MenuPass Group): New Tools, Global Campaign Latest Manifestation of Longstanding Threat, FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/04/apt10_menupass_grou.html.

[7] N. Carr, Cyber Espionage is Alive and Well: APT32 and the Threat to Global Corporations, FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/05/cyber-espionage-apt32.html.

[8] J. O'Leary, J. Kimble, K. Vanderlee, and N. Fraser, Insights into Iranian Cyber Espionage: APT33 Targets Aerospace and Energy Sectors and has Ties to Destructive Malware, FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/09/apt33-insights-into-iranian-cyber-espionage.html.

[9] W. Mercer and P. Rascagneres, Korea In The Crosshairs, Talos Intelligence. [Online]. Available: https://blog.talosintelligence.com/2018/01/korea-in-crosshairs.html.

[10] E. Middelesch, Anonymous and Hidden Communication Channels: A Perspective on Future Developments, thesis, 2015.

[11] R. A. Kemmerer, A practical approach to identifying storage and timing channels: twenty years later, IEEE. [Online]. Available: https://ieeexplore.ieee.org/document/1176284/.

[12] S. Zander, G. Armitage, and P. Branch, A Survey of Covert Channels and Countermeasures in Computer Network Protocols, IEEE Communications Surveys, vol. 9, no. 3, 2007.

[13] Operation Blockbuster: Unraveling the Long Thread of the Sony Attack, Novetta. [Online]. Available: https://www.operationblockbuster.com/wp-content/uploads/2016/02/Operation-Blockbuster-Report.pdf.

[14] B. Lampson, A Note on the Confinement Problem, Commun. ACM, vol. 16, no. 10, Oct. 1973, pp. 613-615

[15] C. G. Girling, Covert Channels in LANs, IEEE Trans. Software Engineering, vol. SE-I3, no. 2, Feb. 1987, pp. 292-296.

[16] National Computer Security Center, US DoD, Trusted Computer System Evaluation Criteria, Tech. Rep. DOD 5200.28- STD, National Computer Security Center, Dec. 1985, http://csrc.nist.gov/publications/history/dod85.pdf.

[17] G. J. Simmons, The Prisoners Problem and the Subliminal Channel, in Proceedings of Advances in Cryptology (CRYPTO), pp. 5167, 1983.

[18] Liping Ji, Wenhao Jiang, and Benyang Dai, A novel covert channel based on length of messages, International Conference on e-Business and Information System Security, 2009.

[19] V. Berk, A. Giani, G. Cybenko, N. Hanover, ”Detection of covert channel encoding in network packet delays”, Technical Report TR536, Dartmouth College. Nov. 2005.

[20] S. Gianvecchio, H. Wang Detecting covert timing channels: an entropy-based approach CCS07: Proceedings of the 14th ACM Conference on Computer and Communications Security, ACM, New York, NY, USA (2007), pp. 307-316.

[21] H. Zeng, Y. Wang, W. Zu, J. Cai, and L. Ruan, New denition of small message criterion and its application in transaction covert channel mitigating, Journal of Software, vol. 20, no. 4, pp. 985996, 2009.

[22] J. Wu, Y. Wang, L. Ding, and X. Liao, Improving performance of network covert timing channel through Huffman coding, Mathematical and Computer Modelling, vol. 55, no. 1-2, pp. 6979.

[23] M. Hussain and M. Hussain, A High Bandwidth Covert Channel in Network Protocol, International Journal of Advanced Science and Technology, vol. 30, May 2011.

[24] N. Pappas, Network IDS & IPS Deployment Strategies, SANS. [Online]. Available: https://www.sans.org/reading-room/whitepapers/intrusion/network-ids-ips-deployment-strategies-2143.

[25] J. Grunzweig, M. Scott, and B. Lee, New Wekby Attacks Use DNS Requests As Command and Control Mechanism, Palo Alto Networks. [Online]. Available: https://researchcenter.paloaltonetworks.com/2016/05/unit42-new-wekby-attacks-use-dns-requests-as-command-and-control-mechanism/.

[26] ”APT32 Un-usual Suspects, FireEye. [Online]. Available: https://content.fireeye.com/apt/rpt-apt38.

[27] Aeris 2.1 User Guide, CIA. [Online]. Available: https://wikileaks.org/vault7/document/Aeris-UsersGuide/Aeris-UsersGuide.pdf.

[28] At the Center of the Storm: Russias APT28 Strategically Evolves its Cyber Operations, FireEye. [Online]. Available: https://www.fireeye.com/current-threats/apt-groups/rpt-apt28.html.

[29] En Route with Sednit Part 2: Observing the Comings and Goings, eset. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part-2.pdf.

[30] Cobalt Strike: Advanced Threat Tactics for Penetration Testers, Cobalt Strike. [Online]. Available: https://www.cobaltstrike.com/downloads/csmanual312.pdf.

[31] A. Dahan, Operation Cobalt Kitty: A large-scale APT in Asia carried out by the OceanLotus Group, CyberReason. [Online]. Available: https://www.cybereason.com/blog/operation-cobalt-kitty-apt.