

Detecting Anomalies in TLS Traffic Using Encrypted Traffic Analysis

Pekka Lupari

Master's thesis

May 2021

Information and Communication Technology

Degree Program in Cyber Security

Author(s) Lupari, Pekka	Type of publication Master's thesis	Date May 2021 Language of publication: English
	Number of pages 94	Permission for web publication: x
Title of publication Detecting Anomalies in TLS Traffic Using Encrypted Traffic Analysis		
Degree programme Information and Communication Technology, Master's program in Cyber Security		
Supervisor(s) Rantonen Mika, Luostarinen Hannu		
Assigned by Cinia Oy, Majuri Pasi		
Abstract <p>Subject of research was assigned by Cinia Oy. Cinia is a Finish company providing network, software and cyber security services and solutions. Subject was delineated to detection of anomalies in Transport Layer Security (TLS). Mitigation, response and forensics activities were out of scope.</p> <p>Primary objective was to gain understanding of methods and tools how one can detect anomalies in TLS encrypted traffic without decrypting it and how opensource products could be utilized. Another, more practical, objective was to investigate how two different commercial products, SensorFleet and LogPoint, could be combined and utilized as an Encrypted Traffic Analysis (ETA) solution. Opensource ETA solution, based on Security Onion and RITA systems, was used as a reference.</p> <p>Research was made using qualitative methods and analyzing was ongoing process during the whole research. First, theoretical data from books, research papers, web articles and videos were analyzed by researcher and then applied in testing phase. Results from tests was analyzed and followed by conclusions.</p> <p>Introduction chapters introduces subject and reasons why subject was chosen. In next chapter, TLS protocol was described focusing on parts important for methods used ETA. In chapter 3, threats hidden inside TLS traffic were described and how they can be detected. Chapter 4 focused on threat detection in general and concepts of Network Detection and Response (NDR) and threat hunting. Chapter 5 introduced opensource and commercial tools and systems suitable for ETA. Chapter 6 described implementing and testing phase of the research. Results was analyzed in chapter 7 and followed by conclusions in chapter 8. Chapter 9 summarized research with discussion.</p>		
Keywords/tags (subjects) ETA, NDR, RITA, Security Onion, SensorFleet, Suricata, TLS, Zeek		
Miscellaneous (Confidential information)		

Contents

1	Introduction	10
1.1	Objectives	11
1.2	Research Methods.....	11
2	Transport Layer Security (TLS).....	12
2.1	Cryptography and Ciphersuites.....	14
2.1.1	Asymmetric Encryption, Keys and Authentication	14
2.1.2	Symmetric Encryption Ciphers	16
2.1.3	Message Authentication Codes (MAC)	18
2.1.4	Pseudorandom Function (PRF)	18
2.1.5	TLS ciphersuites	18
2.2	Certificates and Public Key Infrastructure.....	19
2.3	Handshake Protocol	22
3	Threats in TLS	27
3.1	Malware Types	27
3.1.1	Botnets.....	28
3.1.2	Ransomware	28
3.1.3	Remote Access Trojans (RATs).....	28
3.2	Attack Techniques	29
3.2.1	Phishing.....	29
3.2.2	Data Exfiltration	29
3.2.3	Brute Force	30
3.2.4	Distributed Denial of Service (DDoS).....	30
3.2.5	Scanning.....	31
3.3	Organization Policy Violations.....	31
3.3.1	Tor.....	32

	2
3.3.2 Virtual Private Network (VPN).....	33
3.3.3 DNS over HTTPs (DoH).....	33
3.3.4 TLS Misconfigurations.....	34
4 Threat Detection	34
4.1 Pyramid of Pain	34
4.1.1 Hash Values.....	35
4.1.2 IP Addresses.....	35
4.1.3 Domain Names	36
4.1.4 Network Artifacts.....	36
4.1.5 Tools.....	36
4.1.6 Tactics, Techniques and Procedures (TTPs).....	37
4.2 Network Detection and Response (NDR).....	37
4.2.1 Packet Capturing.....	39
4.2.2 Collecting Flow Data	39
4.2.3 Ingesting, Filtering, Parsing and Forwarding.....	40
4.2.4 Enrichment and Threat Intel.....	41
4.2.5 Rule-Based Detection	41
4.2.6 Behaviour-Based Detection	42
4.2.7 Responses and Integrations.....	43
4.3 Threat Hunting	43
5 ETA Tools and Solutions.....	43
5.1 ETA Tools	44
5.1.1 Suricata	44
5.1.2 Zeek.....	45
5.1.3 RITA.....	48
5.1.4 LogPoint	48

	3
5.2 ETA Solutions.....	49
5.2.1 Security Onion	49
5.2.2 SensorFleet	51
6 Implementation and Testing.....	53
6.1 Implementing Testing Environment.....	55
6.1.1 Implementing ETA1	55
6.1.2 Implementing ETA2	57
6.1.3 Implementing Data Sources	58
6.2 Test Cases	58
6.2.1 Detecting Traffic to Phishing Site	59
6.2.2 Detecting Metasploit HTTPS Reverse Shell traffic.....	61
6.2.3 Detecting SNIcat C2 traffic.....	65
6.2.4 Detecting Tor	68
6.2.5 Detecting DoH.....	71
6.2.6 Detecting TLS Misconfigurations	72
6.3 Administrative and Operational Evaluation	74
7 Research Results.....	75
7.1 Test Results.....	75
7.2 Evaluation of Research Objectives	80
8 Conclusions	80
9 Discussion	82
References.....	84
Appendices	90

Figures

Figure 1. SSL Labs statistics on SSL an TLS versions (SSL Labs 2021)	14
Figure 2. TLS 1.2 full handshake	22
Figure 3. TLS 1.2 abbreviated handshake	24
Figure 4. TLS 1.3 full handshake	26
Figure 5. Tor session establishment.....	32
Figure 6. Pyramid of Pain	35
Figure 7. SOC Triad	37
Figure 8. Example of NDR system architechture.....	38
Figure 9. Packet capture engine in Security Onion	50
Figure 10. Packet capture engine in SensorFleet.....	52
Figure 11. Testing environment	53
Figure 12. Components of ETA systems in NDR architecture	54
Figure 13. SensorFleet IDS Policy Manager configuration for external Suricata rules	55
Figure 14. SensorFleet IDS Policy Manager configuration for Zeek JA3 script	56
Figure 15. SensorFleet Sensor interface configuration for Zeek instrument.....	57
Figure 16. TLS traffic flows to phishing sites and RAT/C2 server	59
Figure 17. SensorFleet IDS Rule Manager Blacklists configuration.....	60
Figure 18. Blacklisted certificate hash detected in LogPoint	60
Figure 19. Blacklisted domain name detect in LogPoint.....	61
Figure 20. Blacklisted IP address detected in LogPoint.....	61
Figure 21. Security Onion alert for JA3 match for malware.....	62
Figure 22. Investigating alert in Security Onion’s Hunt portal.....	63
Figure 23. Querying JA3 hashes from ja3er.com database.....	63
Figure 24. Log entry for Metasploit HTTPS remote shell in LogPoint.....	64
Figure 25. TLS connection in LogPoint indicating possible data theft	64
Figure 26. IoCs in RITA for Metasploit HTTPS remote shell	65
Figure 27. Security Onion Suricata alert for blaclisted certificate subject.....	66
Figure 28. Detecting SNIcat C2 traffic in Security Onion.....	67
Figure 29. Detecting SNIcat C2 traffic in LogPoint	67
Figure 30. Detecting SNIcat beaconing in RITA	68

Figure 31. Investigating Tor Circuit and external IP in Tor browser	68
Figure 32. Security Onion alert for known Tor node	69
Figure 33. IoCs for Tor traffic seen in Security Onion	70
Figure 34. LogPoint Sankey chart for Tor traffic	70
Figure 35. Security Onion Suricata alert for DoH to cloudflare.com	71
Figure 36. IoCs for Cloudflare DoH traffic seen on Security Onion.....	72
Figure 37. Detecting TLS misconfigurations in LogPoint.....	73
Figure 38. Detecting TLS version and ciphersuites in Security Onion.....	73
Figure 39. Detecting x.509 misconfigurations in Security Onion	74

Tables

Table 1. Examples of algorithms used in TLS 1.2 ciphersuites.....	19
Table 2. Fields used in X.509 version 3 certificate	20
Table 3. ClientHello message in TLS 1.2 handshake.....	23
Table 4. Zeek log files useful in ETA	46
Table 5. Resources on VMware platform for SensofFleet	55
Table 6. Resources on VMware platform for Security Onion and RITA.....	57
Table 7. Ease of implementation, administration and operation of ETA solutions.....	76
Table 8. Detecting IoCs for Phishing Site.....	76
Table 9. Detecting IoCs for Metasploit HTTPS reverse shell and data theft.....	77
Table 10. Detecting IoCs for SNIcat C2 traffic	78
Table 11. Detecting IoCs for Tor traffic	78
Table 12. Detecting IoCs for DoH traffic.....	79
Table 13. Detecting IoCs for TLS misconfigurations and policy violations.....	79

Terminology

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-GCM	Advanced Encryption Standard Galois Counter Mode
AI	Artificial Intelligence
ASN.1	Abstract Syntax Notation One
ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
APT	Advanced Persist Threat
Base64	Binary-to-text encoding scheme used in programming
C2	Command and Control
CA	Certification Authority
CBC	Cipher Block Chaining
ChaCha20	Stream cipher mode
CLI	Command Line Interface
CRL	Certificate Revocation List
CSR	Certificate Signing Request
CSV	Comma Separated Values
DER	Distinguished Encoding Rules
DDoS	Distributed Denial of Service
DH	Diffie-Hellman
DHE	Ephemeral Diffie-Hellman
DN	Distinguished Name
DNS	Domain Name System

DoH	DNS over HTTPS
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECH	Encrypted Client Hello
ECDH	Elliptic Curve Diffie-Hellman
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
EDR	Endpoint Detection and Response
ETA	Encrypted Traffic Analysis
ESNI	Encrypted Server Name Indication
EVE	Extensible Event Format
GUI	Graphical User Interface
HMAC	Hash-based Message Authentication Code
HKDF	HMAC-based Key Derivation Functions
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IPSec	Internet Protocol Security
IoC	Indication of Compromise
IoT	Internet of Things

IANA	Internet Assigned Numbers Authority
IV	Initialization Vector
JA3	TLS fingerprinting technique
JSON	JavaScript Object Notation
LXC	Linux Containers
MAC	Message Authentication Code (cryptography)
MAC	Media Access Control (ethernet)
ML	Machine Learning
MRI	Magnetic Resonance Imaging
NGFW	Next-Generation Firewalls
NDR	Network Detection and Response
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
OISF	Open Information Security Foundation
OSCP	Online Certificate Status Protocol
PCAP	Packet Capture
PEM	Privacy-Enhanced Mail
PFS	Perfect Forward Secrecy
PKI	Public Key Infrastructure
PoC	Proof of Concept
PoS	Point of Sales
PRF	Pseudorandom Function
RA	Registration Authority
RAT	Remote Access Trojan

RBAC	Role Based Access Control
RC4	Rivest Cipher 4
RFC	Request For Comments
RSA	Rivest Shamir Adleman
SCADA	Supervisory Control And Data Acquisition
SIEM	Security Information and Event Management
SOC	Security Operation Center
SHA2	Secure Hash Algorithm 2
SHA3	Secure Hash Algorithm 3
SNI	Server Name Indication
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
TLS	Transport Layer Security
TTPs	Tactics, Techniques and Procedures
UEBA	User and Entity Behavior Analytics
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
XOR	Exclusive OR (logical operator)
X.509	Certificate standard
YAML	YAML Ain't Markup Language

1 Introduction

Nowadays most of the browsing traffic seen on internet is being encrypted using TLS (Transport Layer Security) protocols. This is a good thing for data privacy of typical end user but on the other hand it also provides attackers and malware softwares a legitimate protocol to hide their intentions and left undetected. (ENISA 2019, 7)

One way organizations can detect malicious traffic inside TLS is to decrypt it. However, this is against the base ideology of TLS since it breaks the end-to-end confidentiality and integrity of client-server traffic and would be especially problematic for sensitive data, such as banking traffic (ENISA 2019, 11). Decryption of TLS traffic requires also investments on powerful hardware and increase costs for organizations (ENISA 2019, 24). In some countries legislation may not even allow decryption of TLS traffic (Farrel 2018, 7). Latest version of TLS (version 1.3) supports PFS (Perfect Forward Secrecy) cipher suites only and makes passive inspection and decryption of TLS traffic impossible. (F5 2020)

Traditional SOC (Security Operation Center) systems such as SIEM (Security Information and Event Management) and EDR (Endpoint Detection and Response) can be used to detect anomalies and misbehaviors of organization's assets. However, if these devices gets compromised, data gathered from them becomes unreliable. NDR (Network Detection and Response) systems are needed to compliment visibility for compromised and unauthorized devices inside organization's network. NDR can detect, for example command and control traffic used by malwares by gathering and analyzing network traffic data even in case when endpoints are compromised and EDR and syslog becomes unreliable. (Oakley 2020)

Many organizations have huge number of end points connected to their networks, which doesn't support installing any additional softwares. These types of devices are Internet of Things (IoT) devices, such as lightbulbs, SCADA systems, medical devices (MRI and X-Ray), surveillance cameras, vending machines and point of sales (PoS) end points, just to name a few. Lateral movement of these devices would remain invisible without NDR solutions. (Kusek 2019)

NDR is relatively new term and it is not scientifically specified or standardized. Gartner defines NDR as a solution that uses primarily machine learning and other non-signature-based techniques to detect suspicious traffic on networks (Gartner 2020). Encrypted Traffic Analysis (ETA) is another loosely defined term used in the field of cyber security. It could be seen as a sub domain of NDR focused only handling the encrypted traffic. While still being heavily relying on machine learning it also uses signature based detection to efficiently identify known Indication of Compromises (IoCs). For example, FlowMon uses JA3 fingerprints to detect known malicious connection based on the patterns and parameters seen on TLS client-server handshakes. (Flowmon 2020)

1.1 Objectives

Primary objective of thesis was to gain understanding of methods and tools how one can detect anomalies in TLS encrypted traffic without decrypting it and how opensource products could be utilized. Another objective assigned by researcher's employer was to investigate how SensorFleet solution combined with LogPoint SIEM solution could be utilized as an ETA solution.

Thesis was delineated only in TLS protocol since unlike many other data encryption transfer protocols, such as IPSec and SSH, it is usually not possible to block the entire protocol in organization's firewalls because of the many legitimate internet services that are relying on it. Focus of thesis was primarily on detecting ongoing threats and secondary on responsive actions. Threat mitigation (before attack) and cyber forensics (after attack) are related to subject but not in scope of this research.

1.2 Research Methods

Subject of the research was not exactly new topic in cyber security but in the past ETA was done by mainly governmental and military organizations. Nowadays technology to perform ETA is available for everyone via opensource communities and many commercial vendors. By approaching the subject from this perspective, researcher decided to use qualitative research methods and tries to find answers to following questions:

- What is Encrypted Traffic Analysis by definition?
- What frameworks and methods can be used in ETA?
- What tools and systems are available?
- What are the benefits of opensource solutions compared to commercial solutions, and vice versa?
- Which solutions are suitable for which type of organizations?

Analyzing was ongoing process during research. First, theoretical data from books, research papers, web articles and videos were analyzed by researcher and then applied in testing phase. Results from tests was analyzed and followed by conclusions.

In chapter 2, TLS protocol is described focusing on parts that are important for methods used ETA. In chapter 3, threats hidden inside TLS traffic are described and how they can be detected. Chapter 4 focuses on threat detection in general and concepts of NDR and threat hunting. Chapter 5 introduces opensource and commercial tools and systems suitable for ETA. Chapter 6 contains implementing and testing phase of the research, where different tools and systems are first implemented into testing environment and then tested against different kinds of threat cases. Results are analyzed in chapter 7, followed by conclusions in chapter 8. Chapter 9 summarizes the research with discussion.

Even though research includes implementing and testing of different tools and systems in practice, it is not a development research since it is not aiming to provide any production ready solution for certain existing environment. However, tools and systems introduced in research can be used as a building blocks for ETA solutions for organization's real business environments.

2 Transport Layer Security (TLS)

In the early nineties, when modern commercial internet started to rise, there was a need for a security protocol since traditional network protocols which internet was built upon, didn't provide any security mechanisms for data traffic traveling around the world. To address this problem, web browser company Netscape started to develop security protocol called SSL (Secure Socket Layer). It's first version was never publicly released but the second version, SSL 2 was released in November 1994 and

was first implemented on Netscape Navigator 1.1 in March 1995. SSL 2 immediately turned out to be unsecure and was rapidly being replaced by SSL 3 in late 1995, which was a completely new design of the protocol. Internet Engineering Task Force (IETF) founded a TLS working group in 1996 to standardize SSL protocol. Web browser competitors Microsoft and Netscape argued on the content of the standard, which slowed the standardization process. In 1999, TLS protocol was standardized in RFC (Request For Comments) 2246. TLS 1.0 was very much the same as SSL 3 and it was used for a long time. In 2003 TLS extensions was released to complement TLS 1.0 but it wasn't until April 2006 when next version TLS 1.1 was released to fix security issues found on TLS 1.0. In August 2008 TLS 1.2 was released adding authenticated encryption and removing all hard coded security primitives from the protocol making it fully flexible. (Ristić 2018, 1-4)

TLS 1.3 was released in August 2018. Its main improvements to TLS 1.2 was to remove support for legacy cipher suites and forcing connections to use dynamic encryption keys. TLS 1.3 also provides more efficient handshake, making initialization of TLS sessions faster. (RFC 8446, 7)

Today, TLS 1.2 and TLS 1.3 are the recommended TLS versions. Versions prior to 1.2 are considered more or less unsecure and support for them has ended or is being ending on most browsers. Based on Qualys SSL Labs' SSL Pulse report for May 2021, TLS 1.2 is being supported on almost all web sites and support for TLS 1.3 is over 40%. Unsecure TLS versions 1.0 and 1.1 are still widely supported on web servers seen on internet but support for SSL protocols have almost existed. On the Figure 1, solid colored bars are statistics from May and faded bars are statistics from April. (SSL Labs 2021)

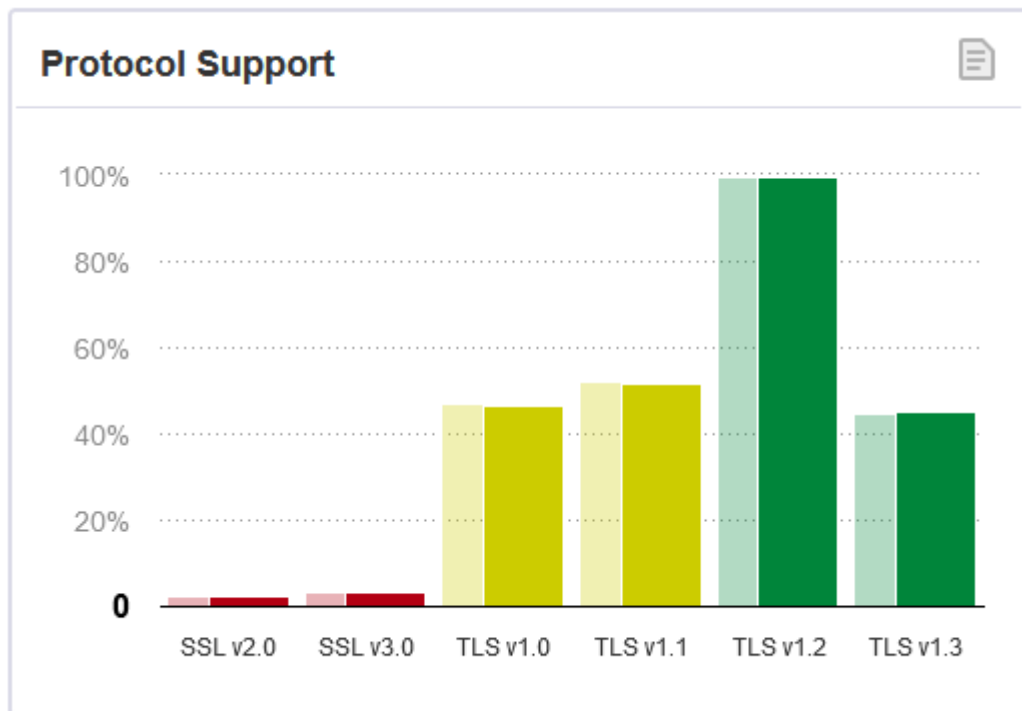


Figure 1. SSL Labs statistics on SSL an TLS versions (SSL Labs 2021)

Since TLS 1.2 and TLS 1.3 are recommended versions and older versions should be avoided, this research focuses mainly on these two latest versions but will also investigate how usage of the legacy versions can be detected.

2.1 Cryptography and Ciphersuites

TLS uses various cryptographic primitives to fulfill confidentiality, authenticity and integrity requirements set for data in transit. Cryptographic primitive is an independent protocol used for specific process in cryptography, for example authenticating data. They can be considered building blocks of cryptographic systems and solutions. (Ristić 2018, 5)

2.1.1 Asymmetric Encryption, Keys and Authentication

Asymmetric encryption uses a key pair for data encryption and decryption. One key, called private key, is used to decrypt data and another key, called public key, is used to encrypt it. The idea is that public key can be easily shared with others who can use it to encrypt and transfer data to owner of the public key. Only the owner of the

public key can decrypt the data using the corresponding private key of the key pair. This provides confidentiality for transferred data. Asymmetric encryption is slow and not suitable for the actual data transfer in TLS. It is therefore used for authentication and negotiation of shared secret keys, which are required for starting symmetric bulk data encryption between client and server. (Ristić 2018, 12-13)

Key exchange is a process used in TLS handshake that relies on asymmetric encryption. TLS support many key exchange algorithms and the one that is going to be used between client and server depends on the support of the algorithm on both sides. There are four main key exchange algorithms: *RSA*, *DHE_RSA*, *ECDHE_RSA* and *ECDHE_ECDSA*. (Ristić 2018, 35-36)

In RSA (Rivest Shamir Adleman) key exchange client generates a 48 byte long premaster secret which it then encrypts using servers public key and sends it as *ClientKeyExchange* message to server. Server then decrypts it with it's private key and gains premaster secret. RSA is simple but has a weakness. Encrypted premaster secret may remain unchanged for years and if some one, who has been recording encrypted traffic, later on gains access to private key, can then decrypt all the recorded traffic. (Ristić 2018, 38)

Diffie-Hellman (DH) key exchange is another protocol used for sharing secret keys between two parties over an insecure connection. Key generation in DH relies on mathematical algorithm containing six variables generated by both client and server and then computing a shared secret based on the variables. Security of the algorithm is based on the assumption that shared key is computationally easy to generate but very hard to reverse back to it's primitives. DH is usually used in Ephemeral mode (DHE), which means that shared secrets are renegotiated for every new session, unlike in RSA. (Ristić 2018, 38-39)

Elliptic Curve DH (ECDH) uses elliptic curve cryptography as an algorithm for generating shared secret. Server sends EC curve type and public point parameter for it to client which then replies with client's public point parameter. Shared secret is calculated based on this information. ECDH is mostly used in ephemeral mode (ECDHE) similarly as in DHE. (Ristić 2018, 40-41)

In TLS, authentication of the server is done during key exchange process. In RSA key exchange, Client generates premaster key signed by server's public key. Server then decrypts it with private key and replies accordingly to client. Since only server can decrypt the message, client has authenticated the server. In DH based key exchanges, server sends shared secret key parameters to client encrypted by server's private key. Since client can decrypt parameters with server's public key, client has authenticated the server. (Ristić 2018, 41-42)

TLS uses three key algorithms called Digital Signature Algorithm (DSA), RSA and Elliptic Curve Digital Signature Algorithm (ECDSA). DSA shouldn't be used since it supports only 1024-bit long keys which are not considered secure enough anymore. RSA is widely used but doesn't scale well when key size increases. If 2048-bit long keys are not considered strong enough, using longer keys might cause performance issues. ECDSA keys are used in elliptic cryptography are superior to RSA keys. 256-bit long ECDSA key provides good security and are fast to process. Generally, 2048-bit long RSA and 256-bit long ECDSA keys are considered secure enough for most use cases. (Ristić 2018, 269-270)

2.1.2 Symmetric Encryption Ciphers

Symmetric Encryption is cryptographic primitive usually used to encrypt application's data in transfer. It uses one key for both encryption and decryption of data. When this key is combined with specific encryption algorithm, also called cipher, one can encrypt clear text data into ciphertext, send it to other party, who can then decrypt it into clear text as long as he or she knows the key and cipher being used. (Ristić 2018, 5-6)

There are two kinds of ciphers, stream and block ciphers. Stream cipher uses keystream, which is infinite long seemingly randomized data. One byte of this keystream is combined with one byte of clear text to produce ciphertext using XOR logic. This process is repeated as long as there is data left to encrypt. Stream cipher are unsecure if keystream gets compromised. This can be avoided by using stream algorithms that derives one-time keys from long-term keys. RC4 (Rivest Cipher 4) is example of a stream cipher. (Ristić 2018, 7)

Block ciphers encrypt data based on specific sized blocks of bytes instead of bytes per bytes as stream ciphers does. Since clear text data is not usually dividable exactly into specific sized blocks, additional padding is needed. In TLS, last byte of encryption block indicates how many bytes is used for padding in the block. AES (Advanced Encryption Standard) is example of block cipher. (Ristić 2018, 8-9)

Block cipher modes are used to encrypt data using specific block ciphers along with other functions to enhance confidentiality and integrity of the encryption. Cipher Block Chaining (CBC) is one of the most commonly used block cipher mode in TLS prior to version 1.3. It uses random string called Initialization Vector (IV) along with XOR function to make ciphertext different everytime even if the clear text would be same. (Ristić 2018, 10-12)

Authenticated Encryption with Associated Data (AEAD) is currently considered to be the best encryption mode available in TLS. It is simpler than CBC since it doesn't need padding or IV. Its functions resembles stream ciphers, but it uses additional 64-bit random value, called nonce, along with ciphertext encrypted using sequence number, header and clear text. (Ristić 2018, 45)

Due to issues found on block ciphers, they are left outside of TLS v1.3 and only AEAD based ciphers are being supported in the latest version of TLS. Unlike TLS 1.2, which supports huge number of various stream and block cipher modes, TLS 1.3 support only five AEAD ciphers: *TLS_AES_256_GCM_SHA384*, *TLS_CHACHA20_POLY1305_SHA256*, *TLS_AES_128_GCM_SHA256*, *TLS_AES_128_CCM_8_SHA256* and *TLS_AES_128_CCM_SHA256*. (Nohe 2019a)

AES-GCM (AES Galois Counter Mode) is one of the AEAD ciphers used for AES based encryption in TLS. It is efficient and secure and is ideal for high-speed data transfers. It was standardized by IETF in August 2008. (RFC 5288, 1)

ChaCha20 was developed to offer an alternative encryption method to AES incase AES algorithm would become vulnerable someday. ChaCha20 generally provides better performance on software-based solutions than AES, which performance relies on AES optimized hardware. It was standardized by IETF in May 2015. (RFC 7359, 1-2)

2.1.3 Message Authentication Codes (MAC)

Hash functions are used to convert data into small fixed size output. Hashes are used to represent and compare large amount of data in compact way. When hash function is reliable, original message can't be computed from the hash and two different messages can't produce same hash. SHA2 and SHA3 are examples of secure hash functions. In TLS, Message Authentication Codes (MACs) are used to authenticate the sender of packets. MAC is a cryptographic function used together with hashing function to encrypt hashes with specific hashing keys. Hash-based Message Authentication Codes (HMACs) are encrypted hashes which can be decrypted only by parties who has the hashing key. (Ristić 2018, 9-10)

TLS 1.3 uses HMAC-based Key Derivation Functions (HKDF), defined in RFC 5689, to verify authenticity of encrypted messages. It is based on extract and expand processes. In extract process key material is first input and then randomized using salting and pseudo random methods to generate pseudo random key. Expanding process then extends the key to required length with additional random data. (Nohe 2019b)

2.1.4 Pseudorandom Function (PRF)

TLS uses pseudorandom functions (PRFs) to generate pseudorandomized data using a secret, a seed, and a unique label. This data is used in generating encryption keys. PRF attribute is mandatory in cipher suites used in TLS versions after 1.2. (Ristić 2018, 48-49)

2.1.5 TLS ciphersuites

In TLS a Ciphersuite specifies collection of algorithms that can be used for authentication and key exchange and to provide confidentiality and integrity of the application data. When initiating a TLS session, client sends it's supported ciphersuites to server. Server then selects one of the ciphersuites and it will be used in client-server TLS connection. If server doesn't support any of the client's ciphersuites, connection is aborted. Ciphersuites in TLS 1.0 to 1.2 are specified using

naming convention

TLS_KeyExchangeAlg_WITH_EncryptionAlg_MessageAuthenticationAlg. (NIST, 14)

TLS 1.2 cipher suites are combinations of different key exchange, encryption and authentication algorithms. *KeyExchangeAlg* consists of two parts, actual key exchange algorithm, such as ECDHE or DHE, and authentication/digital signature algorithm, such as ECDSA or RSA. *EncryptionAlg* specifies algorithm used for symmetric bulk data encryption. *MessageAuthenticationAlg* specifies algorithm used for authenticating the encrypted data using HMACs. There are lots of ciphersuites in TLS 1.2 combining these different algorithms. Table 1 shows examples of algorithms used in TLS 1.2.

Table 1. Examples of algorithms used in TLS 1.2 ciphersuites

Algorithm Type	Algorithm Examples
KeyExchangeAlg	ECDHE_ECDSA, ECDHE_RSA, DHE_RSA
EncryptionAlg	AES_256_GCM, AES_128_GCM, AES_256_CBC
MessageAuthenticationAlg	SHA384, SHA256, SHA

Ciphersuites in TLS 1.3 doesn't specify key exchange algorithm and are presented in simple form *TLS_AEAD_HASH*, where AEAD specifies the AEAD algorithm and HASH specifies HKDF algorithm. There are only five different TLS 1.3 ciphersuites which are presented in chapter 2.1.2. (Nohe 2019b)

Appendix 1 contains list of recommended ciphersuites for both TLS 1.2 and 1.3 versions recommended by National Institute of Standards and Technology (NIST).

2.2 Certificates and Public Key Infrastructure

Public Key Infrastructure (PKI) was developed to solve the problem how people, who have never met, can securely communicate with each other over unsecure networks. In internet PKI this is made possible by using commonly trusted third parties called Certification Authorities (CAs), which issues Certificates used by servers in internet. To provide valid TLS connections on internet, server must generate a Certificate

Signing Request (CSR) and have it signed by one of the commonly trusted CAs. Communications between service provider and CA is done by parties called Subscriber and Registration Authority (RA). After RA have verified that subscriber is who he/she claims to be, CA will sign the CSR using its private key and RA will send the signed certificate back to subscriber. This CA signed certificate can now be validated by each client (Relying Party) who have preinstalled public keys of all the commonly trusted CAs in their root trust stores. Relying Parties can be operating systems, web browsers and other programs. (Ristić 2018, 63-64)

X.509 certificate is a file containing clear text information and public key of the subscriber and digital signature of the certificate issuer. It is used for sharing and storing public keys and is a basic building block of the PKI. X.509 relies on Distinguished Encoding Rules (DER), which is a standard used to encode Abstract Syntax Notation One (ASN.1) defined complex data structures in binary files. Instead of using binary DER files, ASCII formatted PEM (Privacy-Enhanced Mail) files are more popular since the ability to use copy-paste functions with them. PEM is ASCII encoding of DER using Base64 encoding and they can be converted from one to another easily using softwares such as OpenSSL. Most used version of X.509 today is version 3 and it contains certificate fields show on Table 2. (Ristić 2018, 66-70)

Table 2. Fields used in X.509 version 3 certificate

Field Name	Description
Version Number	Version number of the X.509 certificate.
Serial Number	At least 20 bit long randomized number.
Signature Algorithm	Algorithm used for signing the certificate.
Issuer	Distinguished Name (DN) of the certificate issuer, containing at least country, name of organization and organizational unit information.
Validity	Time range which the certificate is valid
Subject	DN of the subscriber
Public Key	Public key algorithm ID, optional parameters and the public key itself

Subject Alternative Name*	Replaces the Subject field and allow binding certificate to multiple instances using of Domain Name System (DNS) names, Internet Protocol (IP) addresses or Uniform Resource Identifiers (URIs).
Name Constraint*	Field which can be used to restrict domains which the CA can issue certificates. Used for subordinate or company owned CAs.
Basic Constraint*	Defines if CA can issue sub CA certificates and the depth of the sub CA path.
Key Usage*	Defines usage for the key, for example Certificate Signer or CRL signer.
Extended Key Usage*	More flexible parameters for defining key usage, for example Server or Client Authentication.
Certificate Policies*	Defines one or more policies the certificate is used for. For example Organizational Identifier.
CRL Distribution Point*	Location where Certificate Revocation List (CRL) can be accessed, for example Uniform Resource Locator (URL).
Authority Information Access*	How to access certain additional information, such as OSCP Responder HTTP URL.
Subject Key Identifier*	Unique value that can be used to identify public key of the subject. Usually hash from the public key itself.
Authority Key Identifier*	Unique value that can be used to identify public certificates of the signing CAs.

*Extensions specific to X.509 version 3 only

Validity of X.509 certificate can be verified using several methods. Checking values seen on fields of the certificate is straight forward process and can easily show things such as is the certificate being self-signed, signed by untrusted CA or if it had become outdated. There are also mechanisms to verify if once valid certification has become revoked.

A Certificate Revocation List (CRL) is a list of serial numbers of certificates that have been revoked. These lists are maintained by corresponding CAs and location where they can be accessed should be stated in the certificate's "CRL Distribution Point field." Online Certificate Status Protocol (OCSP) allows relying parties to query status of specific certificate from OCSP servers, which also are known as OCSP responders. Location of OCSP responder for each certificate should be stated in the "Authority Information Access" field of the certificate. (Ristić 2018, 76)

2.3 Handshake Protocol

TLS uses handshake protocol to establish connection between client and server. Depending on the use case there are six to ten messages exchanged in TLS version 1.2. When client and server establish a new initial connection, a full handshake is made with ten messages as seen in Figure 2.

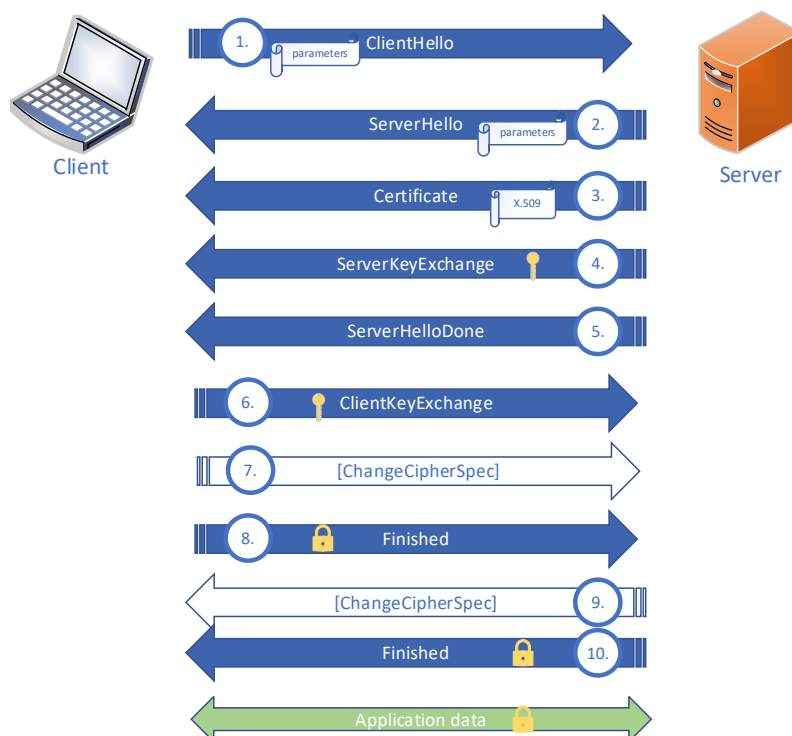


Figure 2. TLS 1.2 full handshake

First, client begins handshake with *ClientHello* message (step 1 in Figure 2). This message contains following fields described in Table 3.

Table 3. ClientHello message in TLS 1.2 handshake

Field Name	Field Value
Version	Highest TLS version client supports
Random	32 bytes of randomized data to make each hello message unique.
Session ID	On initial connection, session ID field is empty. In other cases, it contains 32 bytes of randomly generated data.
Ciphersuites	List of all ciphersuites client supports in order of preference.
Compression	One or more client supported compression methods. Default value is null.
Extensions	contains additional optional data which may be useful in handshake, such as Server Name Indication (SNI), which allows client to specify the name of the server it wishes to connect to.

Server then evaluates message received from client and responds with *ServerHello* message (step 2 in Figure 2). This message contains same fields as in *ClientHello* message but there is only one value inside each field. These are the values which the server has decided to use and client must accept to proceed with the handshake.

Next, server sends its certificate with *Certificate* message (step 3 in Figure 2). This message typically contains server's X.509 certificate chain, where certificates are provided one after another. The exact format and content of the *Certificate* message depends on the ciphersuite which has been chosen. In some case *Certificate* message is not needed at all.

Server also sends *ServerKeyExchange* message (step 4 in Figure 2). This message contains key exchange parameters based on the key exchange defined in ciphersuite. If key exchange parameters from server are not needed, then this message is not sent at all. Finally, server sends *ServerHelloDone* message to notify client that it has sent all intended initial handshake messages (step 5 in Figure 2).

Client evaluates data it has received from server. If TLS version, ciphersuite and certificate are acceptable, client responds with *ClientKeyExchange* message (step 6 in Figure 2). This message contains client's parameters needed for key exchange process. Exact content of the message depends on the key exchange algorithm specified in the ciphersuite. After key exchange process is done and client has generated shared secret, it sends *ChangeCipherSpec* message to server indicating it is ready to start encrypting traffic (step 7 in Figure 2). Finally, client sends *Finished* message indicating that handshake is complete (step 8 in Figure 2). Content of this message is already encrypted using mutually generated shared secret. Encrypted message contains *verify_data* field, which is a hash of all handshake messages to verify the integrity of the entire handshake.

After receiving *ChangeCipherSpec* message from client, server also switches to encryption mode and informs client with *ChangeCipherSpec* message from server (step 9 in Figure 2) and encrypted *Finished* message similarly as client (step 10 in Figure 2). (Ristić 2018, 25-32,58)

When client and server want's restart an old TLS session, they can use an abbreviated handshake consisting only six messages to save clients and servers resources and speeding up initialization of the connection as seen on Figure 3.

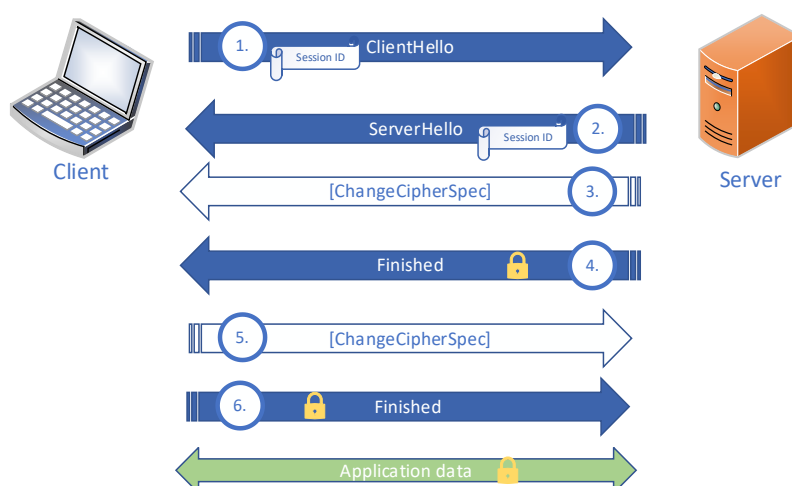


Figure 3. TLS 1.2 abbreviated handshake

Abbreviated handshake starts with *ClientHello* message containing Session ID number of the old session (step 1 in Figure 3). If server is willing to restart old session, it returns the same session ID in the *ServerHello* message, generates new session keys from old master secret key and switches to encryption and sends its *ChangeCipherSpec* and *Finished* message accordingly (steps 2 to 4 in Figure 3).

After receiving messages from server, client does the same thing by starting encryption and finishes handshake (steps 5 to 6 in Figure 3).

Alternative mechanism for restarting old session would be to use session tickets, same way as cookies are used in HTTP connections. Messages are different but handshake process is same as in abbreviated handshake. (Ristić 2018, 34-35)

TLS 1.3 simplifies and makes initial handshake faster compared to TLS 1.2 as seen on Figure 4. Client sends *ClientHello* message containing a random nonce, TLS protocol version, list of ciphersuites, key materials for generating shared secret for various algorithms and potentially additional extensions (step 1 in Figure 4). Server processes *ClientHello* message and determines which ciphersuite to use and then sends *ServerHello* message containing ciphersuite and key exchange material according to chosen ciphersuite (step 2 in Figure 4). After this, server starts to encrypt all messages using symmetric key generated from shared secret. *ServerHello* will be followed by messages containing server's encrypted certificate with optional extensions and *Finished* message to notify client that server is done with handshake (steps 3 to 4 in Figure 4). Client will decrypt messages using symmetric key generated from shared secret key and authenticate server using server's decrypted certificate. Client then starts encrypting messages and sends *Finished* message to server to complete the handshake (step 5 in Figure 4). *Finished* messages contain MAC of the entire handshake. This provides authenticity and integrity of the handshake to both participants. (RFC8446, 9-12)

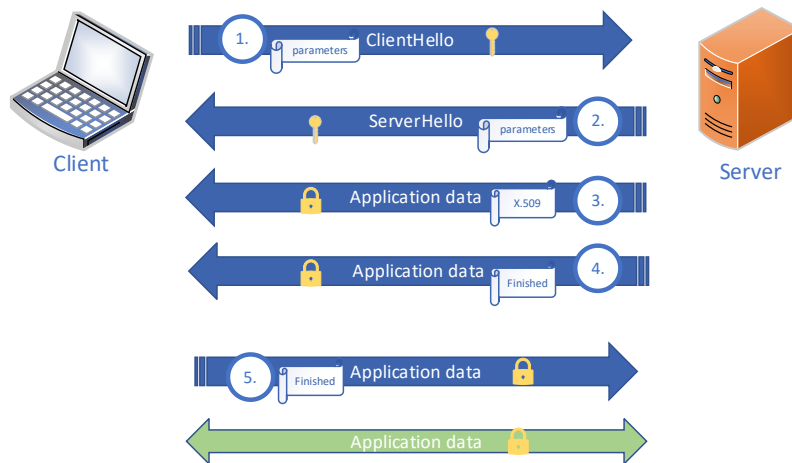


Figure 4. TLS 1.3 full handshake

In TLS 1.3 additional key exchange messages have been removed to speed up the handshake process. Therefore client must generate several keys for various key exchange algorithms and send these inside *ClientHello* message and hope that one of them will be supported and accepted by the server. If server doesn't support any of the key exchange algorithms sent by client, it will reply with *HelloRetryRequest* and handshake starts from the beginning (RFC8446, 13).

TLS 1.3 includes encryption of server certificate but SNI field in *ClientHello* is still being send out unencrypted. This allows eavesdropper to know which web site client is connecting to. Encrypted SNI was developed by Cloudflare in 2018 to address this issue. Idea behind ESNI is that client encrypts SNI field with server's public key before sending it to server. This way only server could decrypt it with its private key. To get server's public key, client must request ESNI key for server during domain name lookup from DNS server. Since this request is usually done in plain text, the initial issue remains and is the main reason why ESNI hasn't been adopted widely.

Encrypted Client Hello (ECH) is another approach to hide sensitive information now visible in *ClientHello* messages. It aims to encrypt all fields in *ClientHello* message. ECH is still under development and not widely used in internet. (Patton 2020)

3 Threats in TLS

In cybersecurity terminology, the word “threat” means any potential danger to organization’s “asset”, which is any physical, virtual or logical item valuable to organization. Threat types includes cyber attacks which aims to destroy or steal organization’s valuable data or disrupt organizations services. Threat actor (TA) types can be categorized to “script kiddies” (nonprofessional people who use existing scripts for hacking), organized crime groups (cyber criminals motivated to make money), state sponsors and governments (governmental agencies motivated to steal intellectual property and perform targeted attacks), hacktivist (hackers motivated by social or political cause) and terrorist (hackers motivated by religious beliefs). (Santos 2020, 9-13)

Besides being used by legitimate applications, such as HTTPS, TLS protocol is also widely used by more questionable applications and by threat actors for their malicious intents. Research made by Sophos Labs indicates that roughly quarter of all malware traffic seen on internet is using TLS for hiding and blending in with legitimate internet traffic (Nagy 2020).

There is also an gray area between legitimate and malicious applications. Applications such as TOR, VPN and DoH (DNS over HTTPS) are not malicious on their own but these tunneling applications can be used to hide malwares and circumvent organization’s security policies. Besides applications inside TLS, the way organization have implemented TLS may pose vulnerabilities due to misconfiguration or use of outdated software and generate a threat targeted to protocol and infrastructure itself.

3.1 Malware Types

Threat actors often use malwares to achieve their goals. Malware is a malicious software designed to cause some sort of harm to target system or victim. It could be for example programmed to encrypt, delete or steal valuable data stored on the system (Kaspersky 2021a). Main malware types utilizing TLS are botnets, ransomwares and RATs.

3.1.1 Botnets

Botnet is a network of hijacked devices (called bots) which are controlled by attacker's server (called bot herder). Bots can be used as a part of another attack, such as phishing, bruteforce or Distributed Denial of Service attacks. Bots receive instructions from bot herder via Command and Control (C2) channel. This communication may be centralized, where there are C2 channels between bot herder and bots, or decentralized, where bots also communicate with each other directly. Decentralized model makes it harder to detect the bot herder (and the attacker itself) and is more commonly used in botnets these days (Kaspersky 2021b). Many botnets rely on TLS to hide their C2 traffic (Desai 2017).

Invalid x509 certificates, traffic to suspicious domains or IP addresses and deviations from baseline TLS traffic are indicators C2 traffic and botnet devices in organization's network.

3.1.2 Ransomware

Ransomware is a malware used to extort money from the victim. After ransomware has been delivered and target system has been infected, it generates encryption keys and starts to encrypt target system's hard drive. After finished it sends the encryption keys to threat actor via C2 channel (usually via TOR network or TLS), deletes local keys and shows an extortion message to victim. Keys used for decryption may (or may not) be delivered to victim after ransom has been paid. (Zimba, Mulenga 2018)

3.1.3 Remote Access Trojans (RATs)

Trojan is a type of malware that is disguised to be a harmless program or file but when executed it does something malicious behind the scenes. Trojans rely on scamming victims to execute them on target system, for example it could be masqueraded as an excel file and delivered to victims via spoofed email from HR. Remote Access Trojans (RATs) are types of trojans that establish a client-server connection from target system to attacker, allowing attacker remote access to target system. Attacker can then do a number of things on the target system, such as browse file system, steal

valuable information (files, password hashes etc.) or simply delete files or destroy systems. (Santos 2020, 18-20)

Highly customized, publicly unknown, RATs are often used by Advanced Persistent Threat (APT) groups usually linked to governmental agencies, for example Chinese governmental APT group has been accused of cyberespionage against corporations, government agencies and other organizations in U.S.A. (McClurg 2020). If RAT is developed and implemented properly, it can be very hard to detect even for the largest organizations. Known RAT application on the other hand are possible to detect using fingerprinting techniques, such as JA3 for Metasploit's TLS based meterpreter session (Althouse 2017).

3.2 Attack Techniques

Threat actors use various techniques to attack their targets. Attacks hidden in TLS include phishing, data exfiltration, brute forcing, denial of service and scanning.

3.2.1 Phishing

Phishing is an attack where targets are usually approached by email and are being scammed to handover valuable information to attacker. Targets may also be lured to download malware into their devices. From TLS perspective, main indicators of phishing attack attempts are links to suspicious sites, usually disguised to look legitimate ones. These fake sites can be identified by verifying that URL is valid against sites such as namecheck.com before entering them. If entering the site, validity of x.509 certification provided by the server should be investigated. Lack of TLS and certificates should be a warning sign. (Pagano 2020)

3.2.2 Data Exfiltration

After attacker has taken control over the target system and found valuable data to steal, he/she would still need to transfer it out of the target system to him/her. Sometimes this can be done using overt communication channels if attacker has no need to hide his/her's activities. For example, attacker may want to get as fast as possible valuable data from target system and then move on to next victim. Another

approach would be to do it discreetly and slowly using covert communication channels. Some RATs utilize backdoors which can be used as covert communication channels for data exfiltration and as an additional remote access to target system. (Santos 2020, 19)

Covert channel usually utilizes legitimate communication channel or protocol in a way it is not intended to be used. For example, protocol header fields can be used to fill with data payload bytes (Santos 2020, 24). In 2020, Mnemonic Labs security researchers discovered a way to exfiltrate data using SNI field of the TLS *client_hello* messages to transfer data payload to server using a tool named *SNIcat*. This exfiltration mechanism bypassed many well-known web proxies and Next Generation Firewalls (NGFW) vendors at the time of testing. (Martrander, Malvica 2020)

3.2.3 Brute Force

Brute force is an attack where login attempt to target system is repeated systematically using trial-and-error method. Brute force attack can be successful if account has weak password (short and without special characters or easy to guess) and target system is using weak authentication settings (not rate limiting login attempts, not using two factor authentication). There are different kinds of brute force attacks. Dictionary attacks targets specific user account and try to login with it using list of popular passwords and words. Reverse brute force attack does the opposite, it goes through list of different accounts against specific password. Credential stuffing is an attack where attacker knows certain valid account and password pair for one system and systematically tries to log into list of other systems with the same credentials. (Kaspersky 2021c)

In TLS, brute force attempts are hard to detect without decrypting and looking into to HTTP messages. However, constant stream of similar packets from same client to server can be indication of brute force attempt, especially if source IP address is suspicious.

3.2.4 Distributed Denial of Service (DDoS)

Distributed Denial of Service is an attack where large number of client devices starts connecting to target system simultaneously. Goal of the attack is to overwhelm the

target system with massive amount of new faked sessions and making it nonresponsive to legitimate requests as well. DDoS attacks are usually made using botnets and the scale of the attack depends on the resources of the botnet. Motivation for DDoS attack can be money (distorting the victim) or to cause outage and harm to certain online service for various reasons. (Kaspersky 2021d)

DDoS attacks can be detected via network monitoring tools and Intrusion Detection System (IDS) systems as a sudden increase of traffic from the baseline.

3.2.5 Scanning

Threat actors often scans target environment networks to gain more information about devices and services before launching the actual attack. Network scan is a kind of scan that sweeps through network address space and tries to identify active hosts and their IP addresses on the network. Port scan is usually targeted to specific host's TCP or UDP ports to gain more detailed information about services and softwares running on these ports. This information can reveal vulnerabilities which threat actors can try to exploit. (Shaw 2020)

Based on researcher experience, scanning attempts coming from internet is very common and usually can be mitigated on organization's perimeter firewall. These scans are many times done by bots and not targeted to any specific organizations and are usually not very interesting from the monitoring perspective. However, scanning activity inside organization network should raise alerts. Network and port scans can be detected via network monitoring tools and IDS system as a sudden increase of connection attempts to many destination IP addresses and TCP/UDP ports.

3.3 Organization Policy Violations

Many times, certain applications are legitimate and acceptable for personal use but when used by organization's endpoints and inside organization network these same applications may pose threats. Same goes to hosted TLS services, what is acceptable in home or testing environments may not be the case in organization network, especially for services accessible from internet.

3.3.1 Tor

Tor network is based on a concept where traffic from client to server is routed through another, encrypted network layer build on top of internet. Tor client software first connects Tor directory server to get list of tor nodes which should be used as a path in tor network to reach the actual target server. Tor client then sets TLS connections with each of the tor nodes specified for path, starting with the entry node. Once initial TLS session is setup, client establish second TLS session to relay node tunneled via initial TLS session. After this, third TLS session is established to exit node, now tunneled via entry node and relay node. Finally, actual client-server TLS connection established, tunneled via all tor nodes in the path. This process is illustrated in Figure 5.

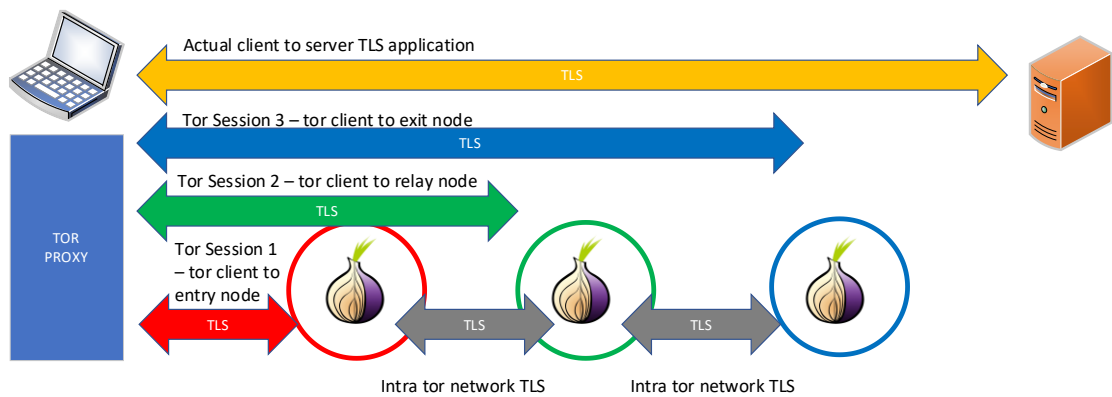


Figure 5. Tor session establishment

This way, tor entry node, and anyone investigating traffic between, knows the client (source IP address) but not the target server (destination IP address) and respectively tor exit node knows only the target but not the client. Unless same party doesn't have visibility into both entry and exit nodes, it is very hard to identify the actual end-to-end client server connection. That is why tor network provides good platform for anonymous activity on internet. (Skerrit 2020)

Tor has legitimate use cases for many individuals but there aren't many reasons why it should be used and allowed in organization's networks. Australian Cyber Security Center (ACSC) recommends blocking tor traffic since it can be used by threat actors

to perform anonymous reconnaissance and exploitation of systems, hide malware C2 traffic and data exfiltration. (ACSC 2020)

3.3.2 Virtual Private Network (VPN)

Virtual Private Network (VPN) softwares and connections are often used for good purpose. Many organizations use them to provide secure connection for remote workers to access organization network over internet. However, VPNs can also be used to bypass organization firewall rules and traffic monitoring, leaving malware and malicious user activity being undetected. Especially TLS based VPNs configured to use TCP port 443 same way as normal web traffic, can be hard to block and detect. (Delaney 2017)

In researcher's opinion, only VPN solution provided by organization's IT should be used and only allowed to connect to specified destinations.

3.3.3 DNS over HTTPs (DoH)

DNS over HTTPs (DoH) is relatively new technology. It has been standardized by IETF in RFC8484 in 2018. It aims to address security issues discovered in original DNS protocol. Instead of performing DNS queries in clear text using UDP protocol, DoH uses TCP, TLS and HTTP as underlying protocols (RFC8484).

In DoH, the main functionality of DNS is still the same, but the client-server connection is now being encrypted using TLS. Web browsers, such as Firefox, can perform DNS queries directly to public DoH servers, bypass endpoint's traditional DNS lookup process and therefore hinder organization's ability to monitor DNS traffic. Since DoH uses TCP port 443 like many other TLS based web applications, it blends in with the rest of the web traffic. Threat actors are taking advantage of this and there are malwares, such as *Godlua*, which uses DoH as C2 channel.

There are certain indicators of usage of DoH in network. Clients browsing the web without traditional DNS requests seen is one indicator. Application fingerprinting techniques, such as JA3, can also be used to detect DoH as well as monitoring connections to well know DoH servers IP addresses. (Hjelm 2019)

In researcher's opinion, DoH should not be used in organization unless there is a specific business need for it, and even then it should be done in a way that organization has visibility and control on it.

3.3.4 TLS Misconfigurations

Referring to researcher's working experience, TLS misconfigurations are not unusual. They occur due to many reasons. Sometimes old server is forgotten and left unpatched running legacy TLS version and ciphersuites. Sometimes new server is running with self-signed certificate, still waiting for CA signed certificate. Sometimes certificates are forgotten to be renewed and run out of date.

There are several ways to take care of these issues. One way is to monitor TLS *server hello* messages seen on organization's network, since it provides lots of information from servers, such as TLS versions and ciphersuites negotiated and certifications fields (only TLS versions prior 1.3). IDS systems, such as Suricata, can be used to monitor TLS traffic and alert for example if TLS versions prior 1.2 are being negotiated, or if invalid certificates are seen. (Suricata 2021a)

4 Threat Detection

Threat detection is a large subject consisting different frameworks, technologies and methods. This chapter explains how framework Pyramid of Pain, Network Detection and Response technology and threat hunting methods can be utilized in threat detection.

4.1 Pyramid of Pain

In 2013 information security professional David Bianco invented a concept called Pyramid of Pain to reflect the different kind of indicators used by APT group called Common Crew. The main idea is that there are several types of indicators an adversary might use to attack organization. These layers form a pyramid where bottom layer is easy for adversary to change and defender to detect. Each layer gets granularly harder while going from down to up, where the highest layer is the most

painful for adversary to change and defender to detect, as seen on Figure 6. (Bianco 2013)

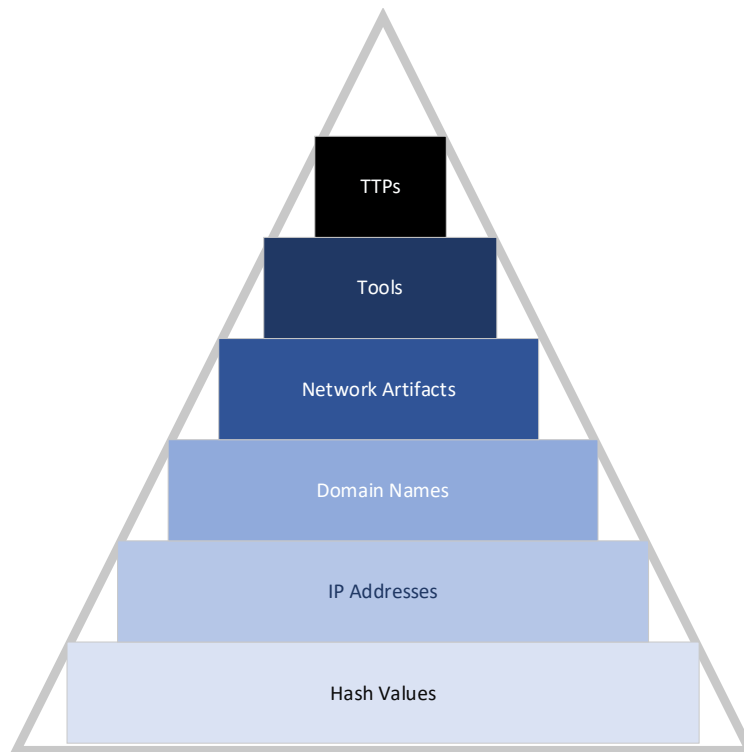


Figure 6. Pyramid of Pain

Pyramid of Pain can be used as a framework for detecting threats in TLS traffic.

4.1.1 Hash Values

Hash values are the easiest ones, since even slightest change done by adversary changes the hash value and avoids defender's hash value-based detection. On the other hand, if defender can detect and match hash value for known bad hash, it is very likely that indicator is a real threat (Bianco 2013). In TLS prior to version 1.3, hash values can be hashes of invalid or blacklisted X.509 certificates and could be detected using CRLs, OCSP and static or dynamic lists from different sources. In TLS 1.3 certificates are encrypted and can't be used as indicators.

4.1.2 IP Addresses

IP addresses can be used as indicators when adversary is attacking remotely. That's why IP addresses are one of the easiest indicators for defenders to detect and block

but also easy for adversary to change (Bianco 2013). In TLS, IP addresses are always present and known bad IP addresses can be detected using static and dynamic IP address blacklists and geolocational information.

4.1.3 Domain Names

Domain names are slightly harder and slower for adversary to change than IP addresses (Bianco 2013). For defender, domain names are good indicators if DNS queries and SNI fields in TLS traffic can be monitored and logged but if protocols like DoH and ESNI are used, domain names are encrypted and not visible for defender. If Domain names can be logged, they can be matched against static and dynamic domain name blacklists.

4.1.4 Network Artifacts

Network artifacts are generally harder to detect than IP and domain name indicators since it requires looking inside application data packets and detect abnormal protocol parameters, such as invalid HTTP user agents or embedded C2 (Command and Control) traffic (Bianco 2013). This layer is tough for defenders when it comes to TLS traffic since application data is being encrypted and cannot be analyzed directly. However, there are certain indicator that can be matched at this layer, such as JA3 fingerprints of TLS handshakes and unnormal traffic behaviors.

4.1.5 Tools

Tools layer includes softwares and applications used by adversary. If these can be detected and blocked, adversary have to redesign and implement his/her's tools or change them completely (Bianco 2013). In TLS, detecting these tools can be very difficult, since application data is encrypted, but certain traffic patterns and behavioral anomalies can be detected by using machine learning algorithms. JA3 fingerprints may identify exact tool, if tool is use unique static parameters in TLS handshake.

4.1.6 Tactics, Techniques and Procedures (TTPs)

On very top layer is the complete attack campaign used by adversary. This includes all the phases adversary uses from reconnaissance of the assets to exfiltration of the valuable data (Bianco 2013). This is the hardest part for defender to find out and will most likely need a human factor to come up with the big picture what the adversary is trying to accomplish based on all the indicators detected on lower layers.

4.2 Network Detection and Response (NDR)

NDR was invented to add additional layer of visibility for SOC teams. SIEM has been widely used to detect threats targeting servers and EDR has been used to detect threats on managed end points. NDR is being used to detect threats in network, such as compromised managed endpoints or unmanaged IoT/OT devices. Using rule based detection and machine learning algorithms, NDR solutions can detect unnormal network activity, such as lateral movement and C2 communications. (Tolbert 2020, 4-8)

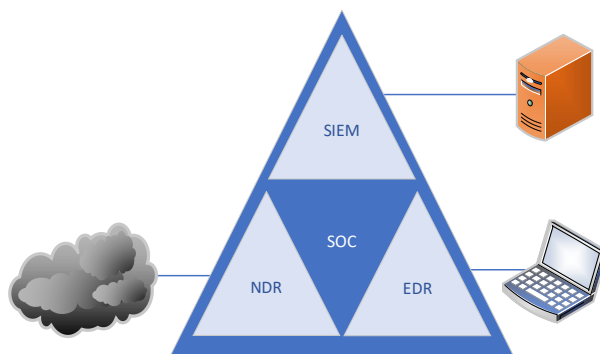


Figure 7. SOC Triad

NDR is relatively new concept and there is no standardized way how to implement it. However, based on researcher's investigations on various NDR solutions there are many similarities how they are designed at high level. There are always external devices (data sources) sending networking data into NDR system, usually by submitting full packet captures or network flow telemetry data. This data is then received by frontend nodes of the NDR systems, often called collectors and sensors,

which are responsible of filtering and preprocessing of data. These nodes then forward parsed and normalized data to NDR centralized components, often called analyzers, which are the core nodes of NDR systems responsible for enriching, analyzing and visualizing data and handling response actions to external systems.

According to Hillstone Networks, NDR solution should include four components: Collection and Storage, Traffic Analytics, Traffic Visibility and Incident Response. Collection and Storage should take care of ingesting, filtering and storing raw data and extracting, parsing and forwarding metadata for Traffic Analytics engine, which will consist of traditional statistical based and machine learning based behavioral analysis. Traffic Visibility should provide security analysts point views, which can be used to drill down in detail to a specific alert, and surface views, which should help to detect traffic anomalies and correlations between different events. Incident Response component is responsible for mitigating and alerting of detected security incidents and enrichment of incidents using external threat intel feeds (Yu 2020). Figure 8 illustrates an example how NDR system functions at high level.

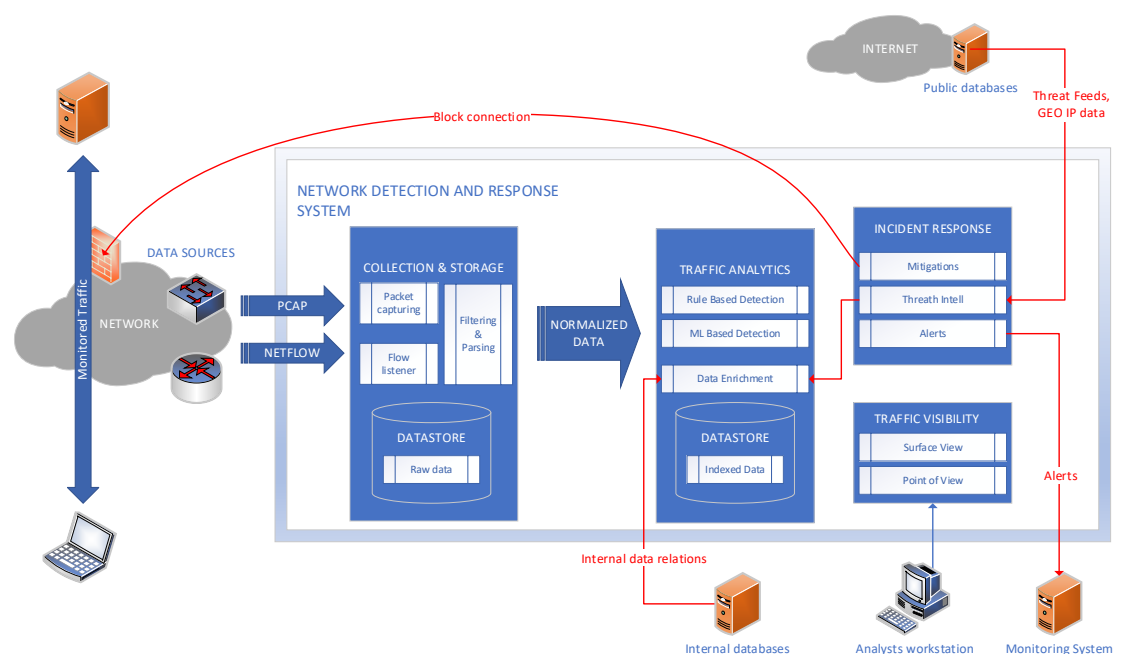


Figure 8. Example of NDR system architecture

4.2.1 Packet Capturing

Packet capturing is a method where full data packets transmitted on the wire are being listened and recorded. This is especially important technique for forensics since the authenticity of the captured traffic is undisputed when collected and stored into dedicated packet capturing system. For example, if threat actor had taken over a computer, authenticity of all data seen on it can be questioned but network traffic leading to incident remains unchanged and reliable since it is stored into external system, unreachable to threat actor. (Messier 2017, 81-82)

Back in the days when hubs were used to connect ethernet devices to each other, everyone connected to same hub could listen everyone, since hubs broadcasts all data packets to all ports. Nowadays, switches have replaced hubs, and only traffic destined to specific device will be sent to it by switch, based on MAC addresses table. Hence, additional tools are needed for bulk packet capturing. One could do physical modifications to copper wires and repeat traffic to capturing device or use specific tools to detect bits transmitted over optical fibers. Port Spanning is method where switch is configured to copy certain ethernet frames to one dedicated port which's only purpose is to transmit these mirrored frames to the wire connected to it. Criteria for mirrored traffic can be based on for example switch's ethernet ports or virtual local area network (VLAN) identifiers, this varies depending on the model of the switch. Some switches don't support port mirroring at all. (Messier 2017, 91-93)

Based on researcher's working experience, port spanning (also called port mirroring) is the most used method in enterprise networks to capture traffic to dedicated packet capturing devices. Port mirroring can usually be applied also in virtual machine environments. In NDR, collectors are responsible for packet capturing.

4.2.2 Collecting Flow Data

Netflow is network telemetry data gathering protocol designed by Cisco. It was initially designed to be used for accounting networking data and to help with bandwidth management. Nowadays it is used also as a network security monitoring tool since it provides non-reputable telemetry data and can be used to detect anomalies. Any netflow enabled network device can be configured to collect netflow

data and forward it to netflow server. With netflow, network itself is acting as a sensor and makes it possible to have wide visibility on what is going on inside network. Netflow creates records of each flows containing at least information of source and destination IP address and source and destination UDP or TCP port and name of the protocol (UDP/TCP). This information is referred as five-tuple and is the basis of all netflow record. Based on the version of netflow protocol and capabilities of network device collecting flow data, there can be also other information gathered from ethernet, IP and transport protocol headers of data flow. Flow records also contain timestamps and traffic counters.

Netflow analyzers can be used to detect threats and traffic anomalies based on collected netflow data. For example, DoS attack can be detected if highly increased number of packets and bytes are seen destined to server and data exfiltration might be occurring if large amount of data is being transferred using abnormal protocol to suspicious destination IP address. Netflow doesn't provide as good insight into traffic as packet capture based solutions does, but it is much more scalable when it comes to implementing data collecting points. Netflow records are much smaller than full packet captures and can therefore be stored much longer which is especially important in forensic analysis. Netflow version 9 is the most used version currently. Older version 5 is also used but other versions are not supported any more. (Santos 2020, 225-237)

IPFIX (IP Flow Information eXchange) is IETF standard based on netflow version 9. It allows network device to be flexibly configured with templates defining which information is being collected. Unlike netflow, relying only on UDP protocol, IPFIX can be configured to transport flow records to server also using SCTP and TCP protocols (Santos 2020, 237-238). Some NDR solutions, such as Cisco Stealthwatch, utilizes netflow data to gain wide visibility through whole network (Santos 2020, 250).

4.2.3 Ingesting, Filtering, Parsing and Forwarding

Data received from packet capture and flow exporters needs preprocessing before forwarding to analyzer components. There is usually some kind of capture engine in collector node, which reads full packets entering Network Interface Card (NIC) and

then forwards them to different tools for further processing. Additional log processing software, such as Logstash or Elasticsearch, can then be used to filter out irrelevant logs, parse logs into suitable format and forward on to NDR analyzer component. (Security Onion 2021)

4.2.4 Enrichment and Threat Intel

Enrichment is a process of adding additional context to normalized data. Log data enrichment is especially useful against IP addresses. It helps security analysts to make decisions when looking alerts and logs, which of them are more likely false positives and which might be true positives and should be inspected with higher priority. For example, log entries of downloaded executables happen constantly. With additional geo or reputational data linked to IP addresses seen on log, analyst can make faster decision if it is false positive or possibly true positive.

There are several sources for data enrichment. External public databases include Geo City (City, state, country IP is registered to) and Geo ASN (organization IP is associated with) lookups. Public DNS and *whois* databases can be used for enrichment as well. Same type of enrichment data can also be gathered from organization's internal databases and linked to organization's IP addresses. (Henderson, Hubbard 2018)

Cyber threat intelligence is information generated from threats seen on the past and present. Threat intelligence information is usually presented as feeds, which contain specific IoCs, such as hashes, IP addresses and domains. Some Threat intelligence feeds are publicly available, and some are private. Private feeds are typically obtained from security vendors against payments. Public feeds are typically provided by opensource communities and governmental organizations. (EC-Council 2021)

4.2.5 Rule-Based Detection

Rule-based detection is usually based on matching specific signature, such as hash, IP address or domain, seen on data packet. This is referred to as signature-based detection and it is efficient for detecting known IoCs. Rule-based detection can also be used to detect known anomalies in traffic, for example a rule can be made to detect series of SYN packet indicated DDoS attack. Weakness in Rule-based detection

is that it can't be used to detect new and unknown threats which doesn't have any signatures available. (Rezek 2020)

JA3 and JA3s are an example of methods used for signature-based detection for TLS traffic. JA3 is a hashing mechanism developed by Salesforce. It uses information gathered from client hello packet during TLS handshake. This information is then used to generate hash value, which can be used to identify applications. Values used for JA3 are picked from fields: TSL Version, Accepted Ciphers, List of Extensions, Elliptic Curves and Elliptic Curve Formats. These values are then inserted into CSV (Comma Separated Values) format and JA3 hash is generated from it using MD5 algorithm. This makes it possible to detect certain client applications due to their unique way to initiate TLS connection. JA3s is similar method used for identifying application running on top of TLS servers. It utilizes values seen on TLS Version, Cipher and Extensions fields of server hello packet seen in TLS handshake and generates JA3s hash using same mechanism as JA3. When used together, JA3 and JA3s hashes have been used to detected malware connection, such as *Metasploit's Meterpreter* and *Cobalt Strike's Beacon*. (Althouse 2019)

4.2.6 Behaviour-Based Detection

Behavior-based detection is method where traffic anomalies are detected using artificial intelligence (AI) and machine learning (ML). This requires collecting and analyzing large amount of data first to generate baseline for normal traffic patterns in environment. Behavior-based detection provides capabilities to detected new threats without fingerprints and known IoCs. (Rezek 2020)

In supervised machine Learning, to ML component is given previously classified data which it uses to learn data classification system. This is very efficient approach for training the system to learn common indicators of certain type known threats, like ransomware, so it could detect similar threats in the future. In cyber security, supervised machine learning is used to train system for previously seen behaviors categorized to be either malicious or benign. New activities are then compared to trained behaviors and labeled based on matched results.

In unsupervised machine learning, there is no pretraining and categorizing of datasets made by human. Unsupervised ML identifies traffic it sees and generates patterns and trends from the data to create baseline for normal traffic using its algorithms. While learning, it constantly detects anomalies from baseline. It can therefore detect zero-day attacks and even threats that no human hasn't imagined yet. (Darktrace 2021)

4.2.7 Responses and Integrations

How NDR system can respond to detected anomalies varies by vendor. Usually, NDR solutions may be able to block detected threat by commanding external systems such as firewalls, routers, switches and servers. Sometimes response can be automatic, sometimes it requires administrative interactions. Some vendors require additional license for automated responses. Generally, all NDR solutions can integrate one way or another to external monitoring systems and forward alerts to them. (Gartner 2020)

4.3 Threat Hunting

Threat hunting offers completely different approach for threat detection compared to NDR. Where NDR is highly automated way to detect ongoing threats, focus on threat hunting is to look for threats that have been left undetected by automation. These kinds of threats might be attacks done by highly skilled APT groups. Threat hunting starts from hypothesis that something malicious which haven't yet been detected might have occurred. Threat hunter then starts to systematically look for evidence to support that theory. Unlike NDR, threat hunting requires human factor and usually threat hunters are experienced cyber security professionals with professional tools. (Chrisander 2020)

5 ETA Tools and Solutions

There are many opensource and commercial tools and systems available which can be used for ETA. This chapter lists few of them. Tools and systems described in this chapter will be also used in testing phase of the research.

5.1 ETA Tools

ETA Tools are softwares designed to do certain specific tasks rather than trying to provide wide range of functionality to cover all aspects needed in ETA.

5.1.1 Suricata

Suricata is well-known and widely used opensource IDS. It inspects network packets using its own rule and signature language. Complex threats can be detected using Lua scripting language. Suricata supports standard YAML (YAML Ain't Markup Language) and JSON (JavaScript Object Notation) formatting and can be therefore integrated with many SIEMs. It is owned by the Open Information Security Foundation (OISF) and is free to use and is widely supported by opensource communities. Suricata is at its best when detecting signature based known threats, policy violations and malicious behavior patterns. It also supports importing rulesets from 3rd party threat intel sources. Suricata is designed to be fast. Single Suricata instance, using multi-threaded code, can inspect several gigabits of traffic. It can identify several protocols, such as HTTP, despised on TCP port it is running, and perform protocol specific inspection rules. Suricata can be used to generate many protocol specific logs and store specific data, such as certificates seen on TLS traffic. (Suricata 2021b)

Suricata rule syntax is simple and easy to read. It includes three parts: action, header and options. The action, determines what happens when the signature matches (for example, drop or alert). The header, defines protocol, IP addresses, ports and direction of the rule. Rule options defines the rule specific parameters, such as message, reference and class type information of the alert.

Suricata is very useful in ETA since it has native support for TLS specific rules. These rules can be used to match many TLS keywords. Appendix 2 shows TLS keywords supported in Suricata version 6. With these keywords, one can for example create a rule which would alert when blacklisted certificate hash is detected using match pattern `"tls.cert_fingerprint; content:"<sha1 hash>";"` or when SNI for blacklisted domain is seen using match pattern `"tls.sni; content:"<domain name>""`

Suricata also has support for JA3 and JA3s fingerprints. Both types can be matched by hash (*ja3.hash; content:"<sha1 hash>";*) or by string (*ja3.string; content:"19-20-21-22";*). keywords for JA3s are similar (*ja3s.hash* and *ja3s.string*).

IP reputation configuration allows Suricata administrator specify IP addresses into categories using reputation files, such as *badhosts.list* and *knowngood.list*. These are CSV files containing values for "ip", "category", "reputation score". "ip" is IPv4 formatted single IP address or network in CIDR notation. "category" is index number defined for reputation category in separate categories file, which is another CSV file listing configured categories. "reputation score" is value from 1 to 127 describing confidentiality of the IP belonging to the specified category.

IP reputation can then be used in rules with keyword "iprep" with syntax "*iprep:<side to check>,<category>,<operator>,<reputation score>*" where first is defined which way traffic is matched (*any, src, dst* or *both*), then the category name, operator (<, > or =) and value of the reputation score. For example, this rule would alert if source IP address belonging to C2 category with reputation greater than 100 is detected: "*alert ip any any -> any any (msg:"IPREP High Value C2"; iprep:src,C2,>,100; sid:1; rev:1;".* Suricata outputs alert using Extensible Event Format (EVE), which is JSON data. (Suricata 2021a)

5.1.2 Zeek

Zeek is an opensource network security monitoring tool. Its development started in 1990s under the name "Bro", but it was renamed to Zeek in 2018. Zeek is a sensor which unobstructively monitors network traffic and generates compact but information rich logs which can be analyzed locally or forwarded to external system such as SIEM. (Zeek 2021b)

Zeek generates wide range of logs based on traffic it sees on the wire. Application specific log files, such as for HTTP sessions, include telemetry data at application level, such as requested URIs and key headers. By default, logs are written in JSON form which can be easily stored into external databases and processed by SIEM products. Zeek also provides built-in functionality to detect and analyze anomalies, for example files can be extracted and compared against external registries to detect

malwares. More complex and customized detection can be done by using Zeek's own scripting language which is similar in nature to Python.

Zeek is designed to handle high-speed network traffics. With proper hardware resources, clustering and load balancing, Zeek can be used even in 100GE networks. Zeek's clustering capabilities provide good scalability as additional Zeek nodes can be easily added to cluster whenever needed. Unlike IDS systems, such as Suricata, Zeek is not optimized for signature detection. Zeek is optimized to interpret network traffic and generate logs with lots of application specific information.

Zeek's architecture is layered into two main components. Event engine is the core of Zeek. It ingests packets seen on the network and generates series of higher-level events based on the packet stream. Events are policy-neutral, simply stating what has been seen on the network. Second main component, Script Interpreter, then process these events based on in-built and custom scripts written in zeek scripting language to detect anomalies, generate logs and raise alerts to external systems.

Even though Zeek can be configured to trigger alerts, it is more suitable for threat hunting or additional investigations when security analyst receives alert from traditional alerting system, such as IDS or EDR. When doing analysis for real time traffic, Zeek ingests network traffic using NICs configured for packet capturing. When doing analysis for offline packet capture data, packet capture files in pcap-format can be imported and processed by Zeek. Zeek creates log files to local storage and can perform log archiving and log forwarding based on configurations. Log files useful in ETA are listed in Table 4.

Table 4. Zeek log files useful in ETA

Log File	Description
conn.log	TCP/UDP/ICMP connections
dns.log	DNS activity
ssl.log	SSL/TLS handshake info
x509.log	X.509 certificate info
intel.log	Intelligence data matches

known_certs.log	SSL certificates
known_hosts.log	Hosts that have completed TCP handshakes
weird.log	Unexpected network-level activity

conn.log contains TCP and UDP flow data, including data captured from packet headers of OSI layer 3 and 4. *conn.log* data is similar to data which can be gathered by netflow. Zeek generates unique connection ID for each entry in *conn.log*, called "*uid*". These "*uid*" values are shared with related application specific log files to ease correlation of separated log files. TLS protocol fields, such as version, ciphersuite negotiated and SNI, can be seen in *ssl.log* files. JA3 and JA3s fingerprints are not included natively in Zeek v4.0.1 but can be installed using Zeek's package manager. JA3 and JA3s hashes are then seen on *ssl.log* aswell. If ESNI or ECH is used, certain fields encrypted by these protocols, such as SNI, are not seen in *ssl.log*.

Another important Zeek log files in ETA is *x509.log*. These logs contain information gathered from field seen on x.509 certificates. Each analyzed certificate is given specific certificate id called "*cert_chain_fuids*". Same field is used in *ssl.log* to ease mapping between ssl and x509 logs. x509 logs contains useful information for ETA such as serial number, subject, issuer and validity of the certification. Since TLS 1.3 encrypts x509 certificates, there are no *x509.log* entries for TLS 1.3 connections.

Zeek has framework called Intel which can be used to feed threat intelligence information to Zeek, such as IP addresses of known threat actors. When Zeek matches this information during traffic analysis, it writes an entry into *intel.log* file. This log has information where specific intel information was seen and which information data it was mapped to. Zeek doesn't support dynamic threat intel feeds directly as data to intel framework must be imported using Zeeks input framework in tab separated ASCII files. Additional mechanism must be used to preprocess threat intel data into correct format and into Zeek's input framework. Zeek supports GeoIP information to enrich log data if Maxmind's *libmaxminddb* package is installed on system. GeoLite2-City database information can also be used to map IP addresses at City level. (Zeek 2021a)

5.1.3 RITA

Real Intelligence Threat Analysis (RITA) is an open source system designed to detect C2 activity on network traffic. It is developed by Active Countermeasures. Instead of trying to detect exact fingerprints of certain IoC, it tries to detect anomalies in traffic behavior. RITA ingests Zeek logs and uses machine learning algorithm to detect beaconing and tunneling activities. It also supports matching domains and IP addresses against blacklists. (RITA 2021)

RITA is written in Golang (Go) programming language. It uses Mongo databases which allows user to isolate different datasets into individual databases when needed. This is defined when importing zeek logs into RITA. (Goddard, N 2020)

RITA uses machine learning algorithms for detecting beaconing. Factors used in beaconing are interval, data size and jitter. Interval defines consistency of heartbeat signals seen on connection. Data size is another aspect to look for in beaconing, if all packets are same sized, it is possibly containing fixed sized C2 commands. Jitter defines how much variation there is seen between intervals. Based on values seen on these factors, beacons are scored and listed in RITA given security analyst information for suspicious connections. Other indications for C2 traffic are long connections, connection to blacklisted IP addresses and domains and detected user agents. (Strand, J 2020)

5.1.4 LogPoint

LogPoint is a commercial SIEM product and not especially designed for ETA.

However, it can be utilized as a component in ETA solution for its capabilities to analyze and visualize log data. LogPoint includes User and Entity Behavior Analytics (UEBA) module, which uses machine learning to detect anomalies and deviations for user and network activities compared to baseline data seen on environment (LogPoint 2021). LogPoint is an example of a tool an organization might already have implemented for another purpose, but which might be used as a component for ETA solution as well. One research aspect assigned by researcher employer was to investigate if LogPoint can be utilized as analyzer in ETA system.

5.2 ETA Solutions

ETA Solutions are larger systems containing different tools interacting with each other. ETA Solutions aims to provide good coverage for all aspects needed in ETA.

5.2.1 Security Onion

Security Onion is free opensource linux distribution designed for threat hunting, security monitoring and log management. It includes wide range of security tools such as Elasticsearch, Logstash, Kibana, Suricata, Zeek and Wazuh. Security Onion started in 2008 and was originally based on Ubuntu but is now container based, not limited to any specific operation system and called Security Onion 2.

Security Onions core functionalities are full packet capture, network and endpoint detection and powerful analysis tools. Packet capturing is done via program called Stenographer, which stores full packet captures to hard drive based on configuration for storage limitation. It has built-in functionality to purge old data and keep disk space available for new packet captures. Full packet captures are important when investigations require exact information what traffic has been seen on network. Network and endpoint detection functionality gathers log and alert data for detected events happening on monitored network and endpoints. For network events Suricata is used to do signature-based detection and Zeek is to gather protocol metadata. Opensource IDS softwares for endpoints such as Wazuh or osquery can be used for monitored endpoints.

Security Onion uses Linux kernel's *AF_PACKET* software to capture packets and load balance to them to another processes utilizing packet capture data, such as network traffic analysis software Zeek as seen on Figure 9. These tools then ingests and processes packet capture data, generating their own log files. For example, Zeek generates different kinds of log files based on protocol seen on packet capture data.

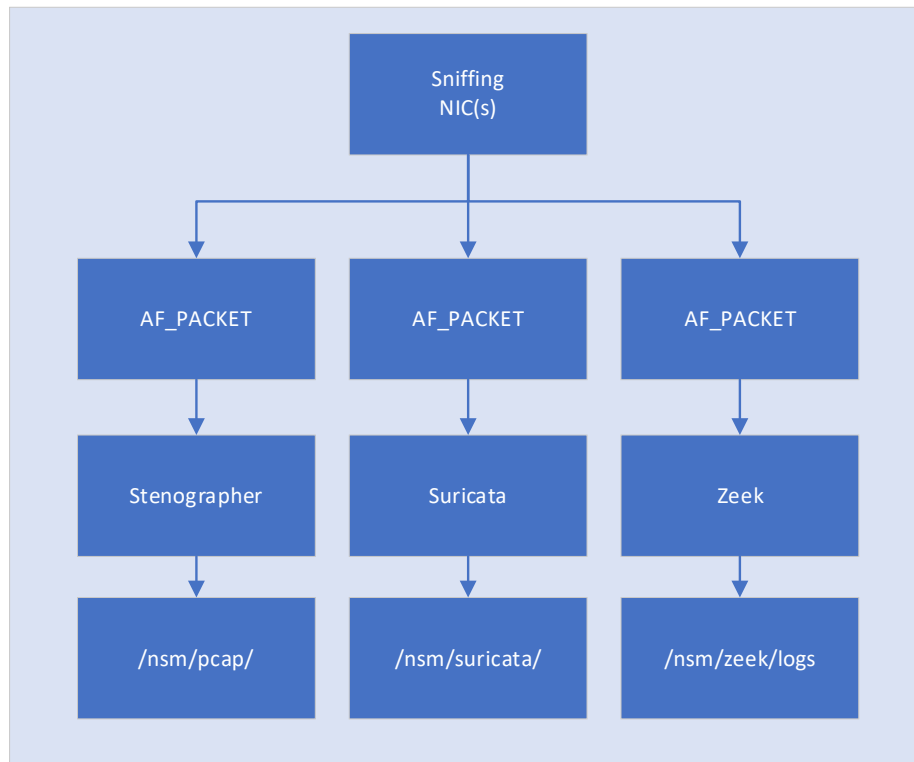


Figure 9. Packet capture engine in Security Onion

Security Onion has GUI which includes separate portals for different functions.

“Alerts” portal allows user to see active alerts generated by network and host IDS.

“Hunt” portal allows user to perform queries from different logs processed by

Security Onion. “PCAP” portal allows user to investigate full data packets gather via Stenographer.

Security Onion can be installed in several ways based on environments and user’s needs. Most simple installation architecture is "Import" where a single node is used without packet capturing capabilities. In "Import" architecture, user can import packet capture data in pcap format manually and then analyze it with Suricata and Zeek and index it by utilizing Elasticsearch. "Evaluation" architecture includes additional NIC for packet capturing traffic from wire. It is designed to be used for testing only and is not recommended for production use. "Standalone" architecture is single node installation which has additional Redis component for queuing logs between Logstash instances. It is suitable architecture for production usage in small environments. "Distributed" architecture contains multiple nodes. Different functionalities are distributed between different types of nodes, for example

separate *Management* node is used for collecting, parsing and storing logs coming from *Forward* nodes which perform packet capturing and log generation. Additional *Search* nodes are used by security analyst for operative tasks.

Components, such as Suricata and Zeek, are run in Docker containers inside Security Onion nodes. Security Onion uses Salt orchestration for managing these containers. Many configurations for containers in Security Onion are done using *pillars*, which are a YAML files used by Saltstack. *Pillar* files can be global or minion. Global *pillar* files all used for making assignments for all Security Onion nodes. Minion *pillar* files are used for node specific configurations. Services can be stopped or restarted in CLI using “*so-<component>-<verb>*” syntax. For example, “*sudo so-zeek-restart*” would allow privileged user to restart Zeek service. (Security Onion, 2021)

Security Onion Solutions, LLC, is a company behind creation and development of Security Onion. It provides support services and training and sales hardware appliances for its customers. (Security Onion Solutions, 2021)

5.2.2 SensorFleet

SensorFleet is a Cyber security sensor solution developed by Finnish company SensorFleet Oy. Its architecture consists of instruments, sensors and manager nodes. Instruments are containerized software components designed to do certain specific tasks. SensorFleet uses opensource instruments, such as Suricata and Zeek, along with tools they have developed by themselves. SensorFleet also supports 3rd party instruments, even user's own developed tools as long as they can be deployed as containers. Instrument are run on sensor nodes, which can be virtual machines or hardware appliances. SensorFleet manager node is designed to be centralized point of managing configurations and deploying instruments to sensors. For example, blacklists, IoCs and rulesets can be configured in manager and pushed from there to sensor platform. SensorFleet aims to ease the pain of managing wide range of tools scattered around many nodes across different networks. (SensorFleet, 2021a)

SensorFleet manager communicates to sensors using VPN. Configurations in YAML format are pushed using ansible playbooks. Sensor and manager nodes have GUI and SensorFleet's own CLI which can be accessed from operation system's CLI using

command "fleet". Both GUI and CLI can be used for configuring sensors.

SensorFleet's centralized architecture consists of Fleet Manager, Sensors and external SIEM solution. Fleet manager is running IDS Policy Manager and Downloader instruments used for managing IDS policies and downloading rulesets from internet. Sensors are running instruments used for detection, such as Capture Engine, Suricata and Zeek. SIEM is the centralized place for analyzing logs generated and forwarded by Sensors.

SensorFleet sensor use its own capture engine to do packet capturing and distribute full packets to different instruments. Capture engine is given dedicated NICs which are configured to receive data and forward it to component called mirror-bridge, which will forward data to instruments attached to it as seen on Figure 10.

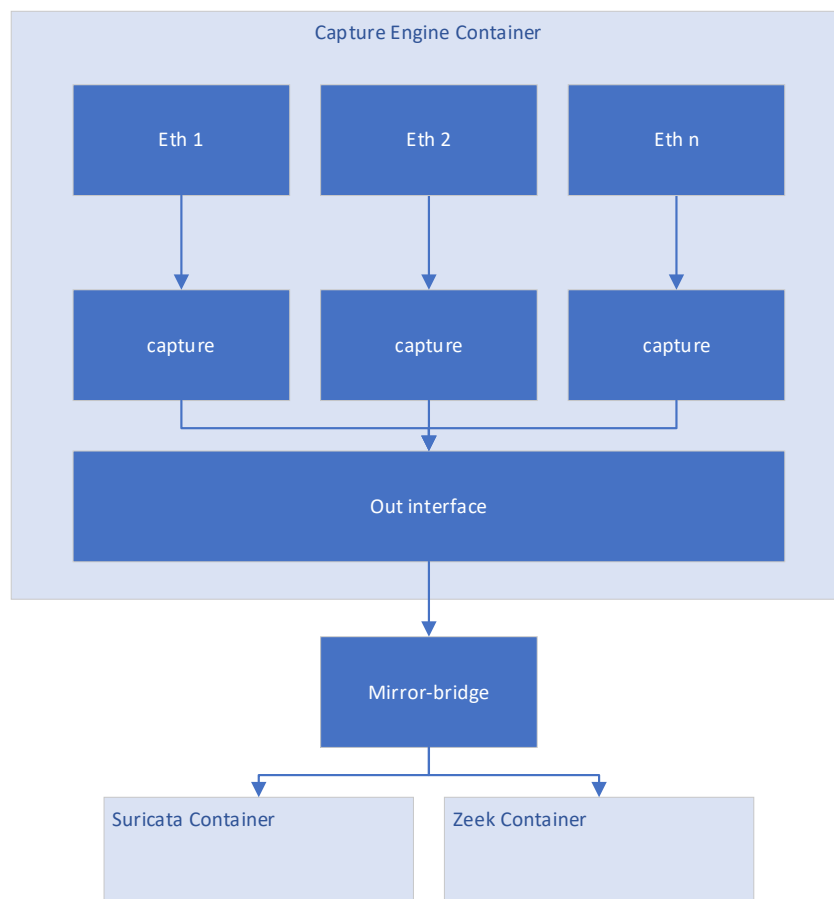


Figure 10. Packet capture engine in SensorFleet

Instruments are run in LXC containers and NICs allocated to them are not accessible from OS. LXC configurations are managed by SensorFleet's own orchestrator software. (SensorFleet, 2021b)

6 Implementation and Testing

Testing was done using Cinia's VMware platform for virtual machines (VM). It was dedicated platform for testing purposes with access to internet to ease installation of virtual machines. On top of VMware platform, virtualized environment was created for imaginary company called Lupari Oy containing separate subnets for workstations and servers segmented by firewall. Additional network was created to simulate internet and to provide connectivity for threat actors targeting the company. ETA systems were implemented and configured to detect anomalies in TLS traffic. Network diagram of the environment is illustrated in Figure 11.

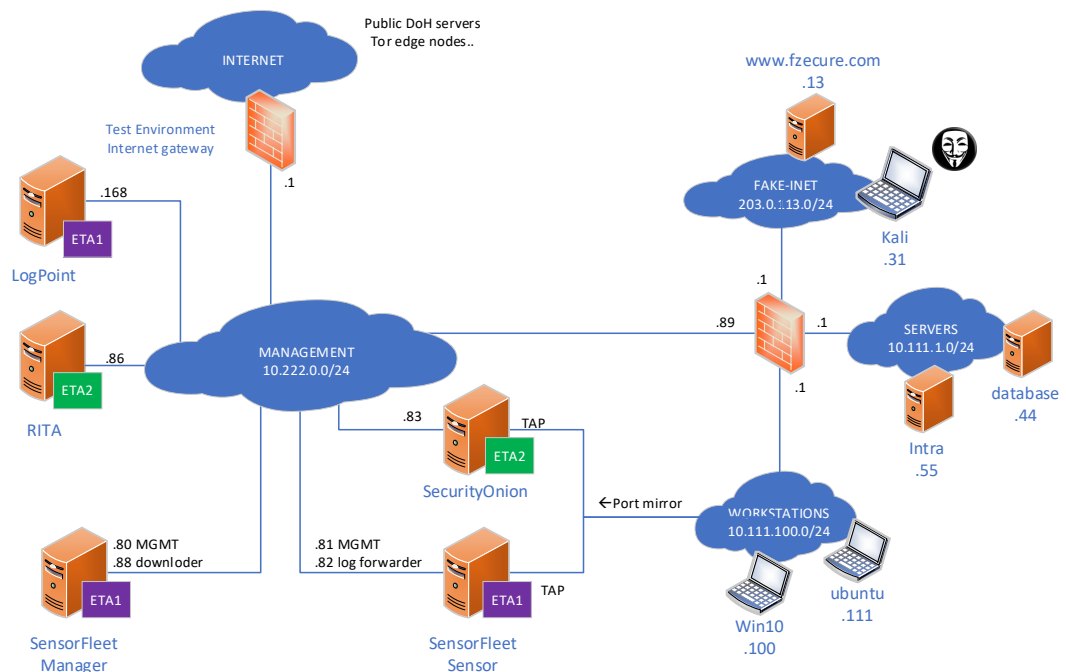


Figure 11. Testing environment

IP addresses used in testing environment were reserved address spaces assigned by IANA (Internet Assigned Numbers Authority) and to be used in private and testing

purposes (IANA 2019). Address space 203.0.113.0/24 was used to simulate internet inside testing environment, 10.111.0.0/16 addresses were used for test organizations internal networks and 10.222.0.0/24 was used for management network for ETA systems. Management network was also used for accessing real internet when needed.

ETA solutions used in testing environment included five virtual servers which created two parallel ETA systems which were compared against each other. ETA1, included SensorFleet manager and sensor virtual machines and LogPoint SIEM solution. ETA2 consisted of SecurityOnion and RITA virtual machines. How components used in ETA systems could be seen in NDR architecture is illustrated in Figure 12.

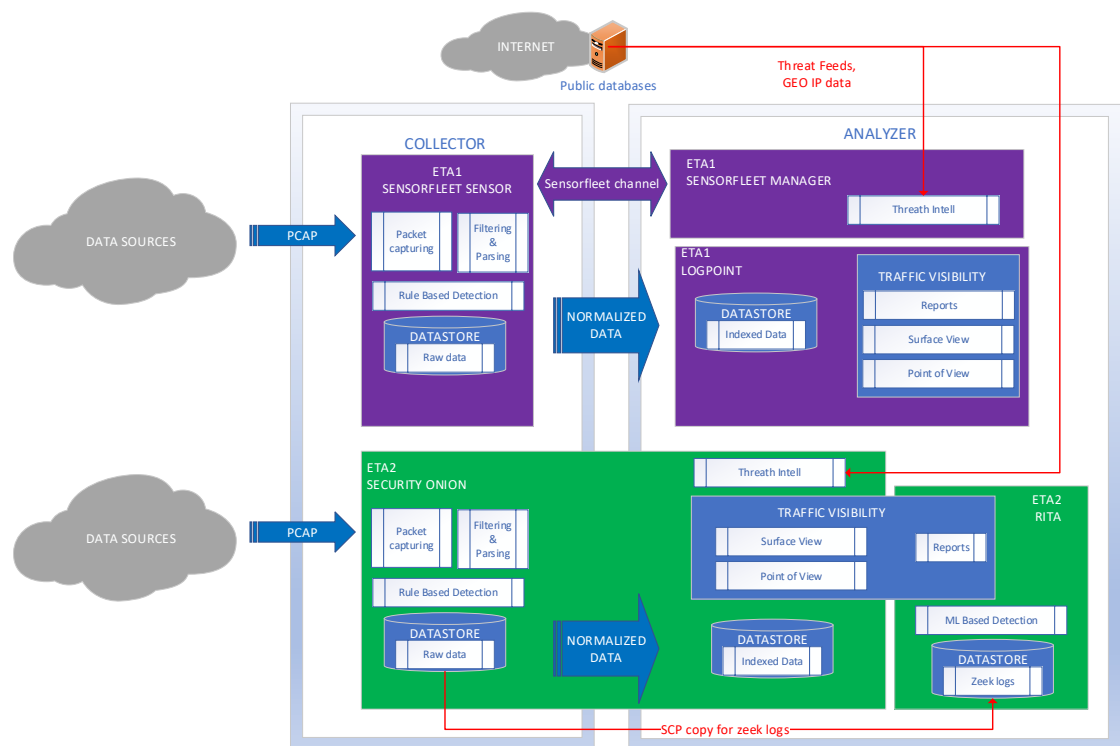


Figure 12. Components of ETA systems in NDR architecture

Responsive part of NDR architecture was out of scope of testing. Both systems have similar collector systems utilizing Zeek and Suricata.

6.1 Implementing Testing Environment

Testing was done on VMware platform which included preinstalled network 10.222.0.0/24 with internet connectivity and LogPoint SIEM. Additional networks and VMs were implemented to build environment for ETA testing.

6.1.1 Implementing ETA1

For ETA1 system two additional VMs were created, SensorFleet manager and sensor with resources listed in Table 5.

Table 5. Resources on VMware platform for SensofFleet

Virtual Machine	CPU	RAM	HDD
SF Manager	2	4 GB	20 GB
SF Sensor	8	16 GB	60 GB

SensorFleet VMs were provided by SensorFleet's representative to be used for testing. Both VMs were running Ubuntu 18.04. SensorFleet version used in testing was 2.3.1. SensorFleet Manager was configured with additional interface to be used for Downloader instrument, which would be used by IDS Policy Manager instrument for retrieving data from internet. IDS rule manager was configured to use dynamic Suricata rules provided by Emerging Threats as seen on Figure 13.

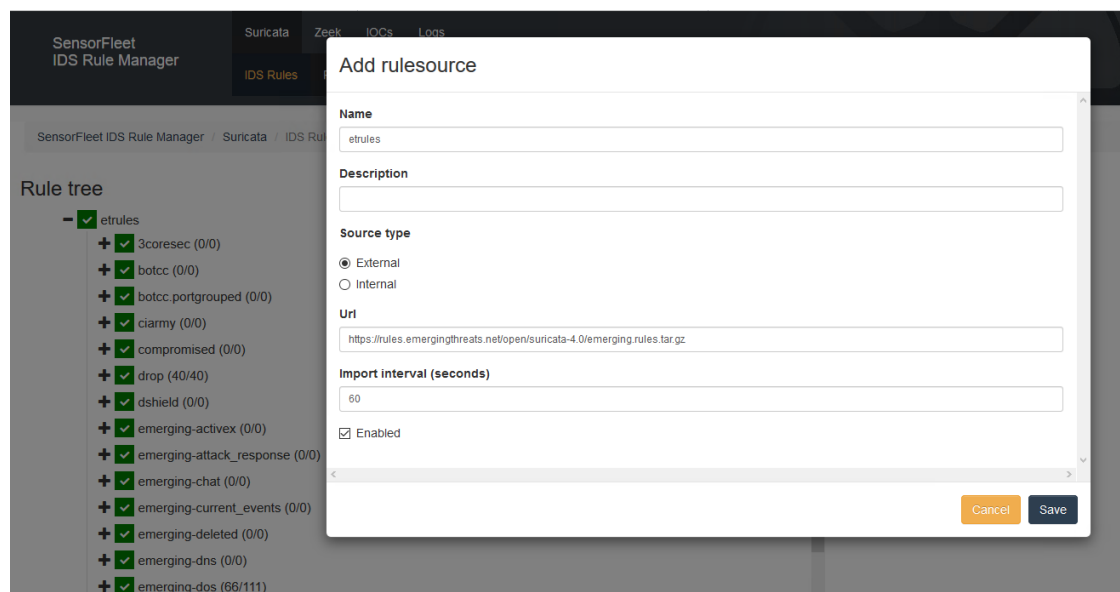


Figure 13. SensorFleet IDS Policy Manager configuration for external Suricata rules

In IDS Policy Manager's Zeek Scripts page, JA3 and JA3s scripts were created as seen on Figure 14.

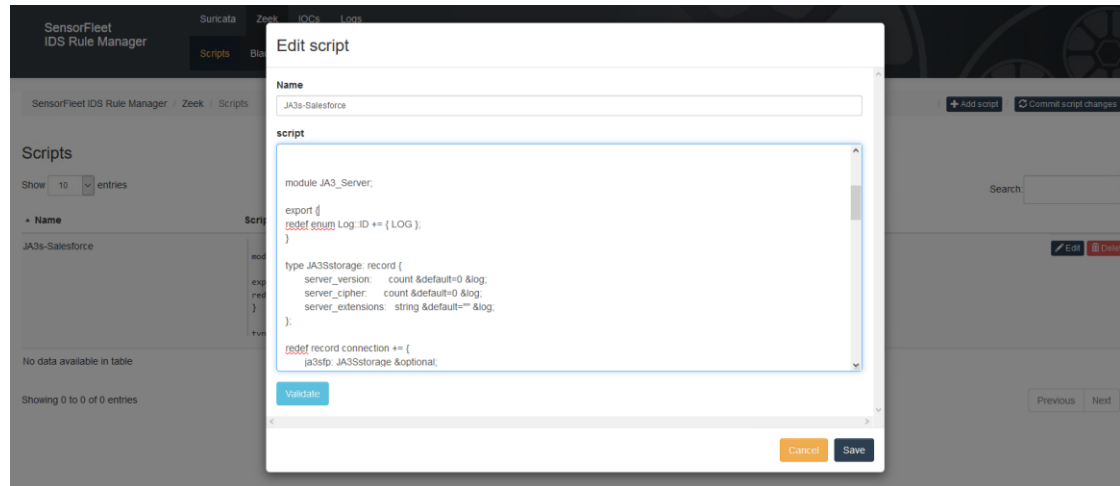


Figure 14. SensorFleet IDS Policy Manager configuration for Zeek JA3 script

SensorFleet sensor VM was configured with two additional interfaces, one for capture engine instrument and one for log forwarder instrument. Instruments for Suricata and Zeek were also added and configured to receive packet capture from mirror-bridge attached to capture engine. Figure 15 show how this was done for Zeek instrument. Configuration for Suricata instrument was similar.

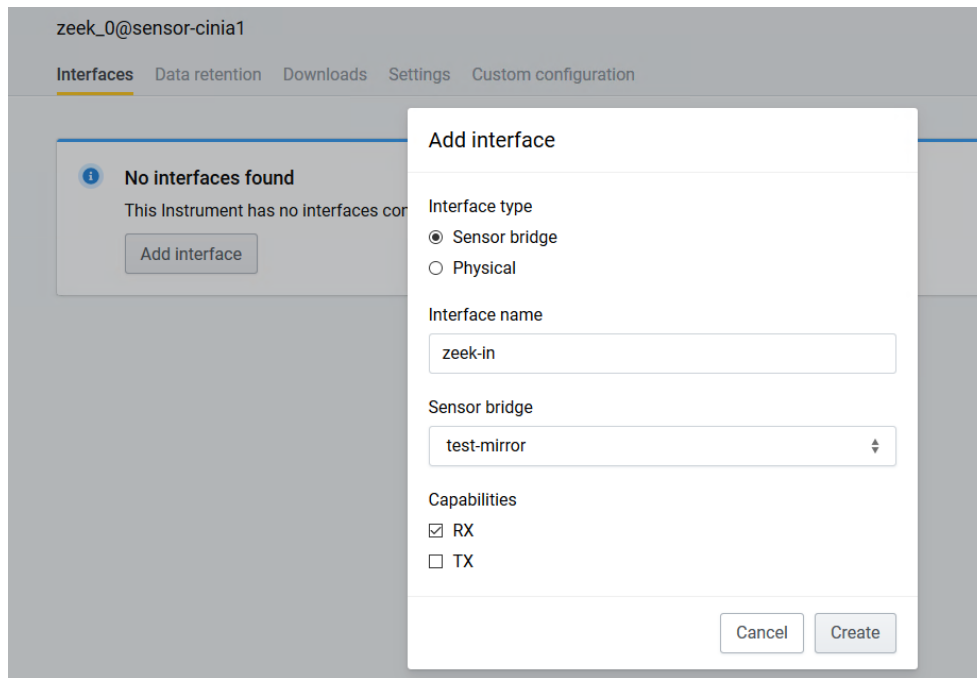


Figure 15. SensorFleet Sensor interface configuration for Zeek instrument

Logforwarder instrument was configured with IP interface and to parse and forward Zeek and Suricata logs to LogPoint. LogPoint was configured to ingest syslogs in JSON format coming from Logforwarder's IP address.

6.1.2 Implementing ETA2

For ETA2 system, two VMs were created, Security Onion and RITA. Resources allocated for these virtual machines are listed in Table 6. Security Onion version used in testing was 2.3.50 and RITA version was 4.2.0 and it was run on top of Centos 7.

Table 6. Resources on VMware platform for Security Onion and RITA

Virtual Machine	CPU	RAM	HDD
Security Onion	8	32 GB	200 GB
RITA	2	16 GB	30 GB

Security Onion was installed using iso file downloaded from Security Onion's Github page. Additional interface was added for virtual machine to be used for packet

capturing. Installation was very straightforward for standalone single node implementation. During installation, basic settings for interfaces, management, Zeek and Suricata were specified by interactive Security Onion Setup program. Overall Installation of Security Onion used in testing was easy but slow. It took couple of hours before Security Onion was up and running after given all the parameters during installation.

Implementing RITA was easy and quick by using installation script from RITA's Github page. Additional filtering and blacklists could have been configured for RITA, but this wasn't done for testing environment as size of the log data wouldn't be that big.

6.1.3 Implementing Data Sources

Several VMs were installed to generate TLS traffic for test cases. There were two VMs placed into "fake internet" and to be used by threat actors. Phishing site was running NGINX web server on Centos 7 VM and Metasploit and SNIcat servers were run on Kali 2021.1 VM. Two servers were installed into company's server network. Both run NGINX web server on Centos 7 VM but with different TLS configuration. Two workstations were placed into company's workstation network, one installed with Windows 10 and one with Ubuntu 18.04. OpenWRT was used for one VM which act as a company's firewall and routed traffic between networks. Additional port mirroring configuration was made in VMware platform which mirrored traffic from workstation network to ETA systems. List of VMs used in testing environment can be seen on Appendix 3.

6.2 Test Cases

Couple of scenarios were tested. First, an attack simulation was played and tested how ETA system could detect it. Simulation starts from the point where attacker has successfully scammed victims via phishing mail to download and install software which are claimed to be antivirus client install packages for their workstations. Attacker is using his web site www.fzecure.com as a place to distribute his malwares. Site has valid certificate signed by CA of the testing environment. When victims have downloaded and run malware packages in their workstations, RAT/C2 client softwares are launched and starts connecting to attacker's server using TLS

connections. After gaining remote access to victim's workstations, attacker performs data theft by uploading large amount of data from workstation to server. All these activities should provide IoCs to ETA systems. Two kinds of malware software were tested. Metasploit's meterpreter over TLS was used for windows workstation and SNIcat was used for Ubuntu workstation. Figure 16 illustrates TLS traffic flows from workstations to phishing site and to attacker's server.

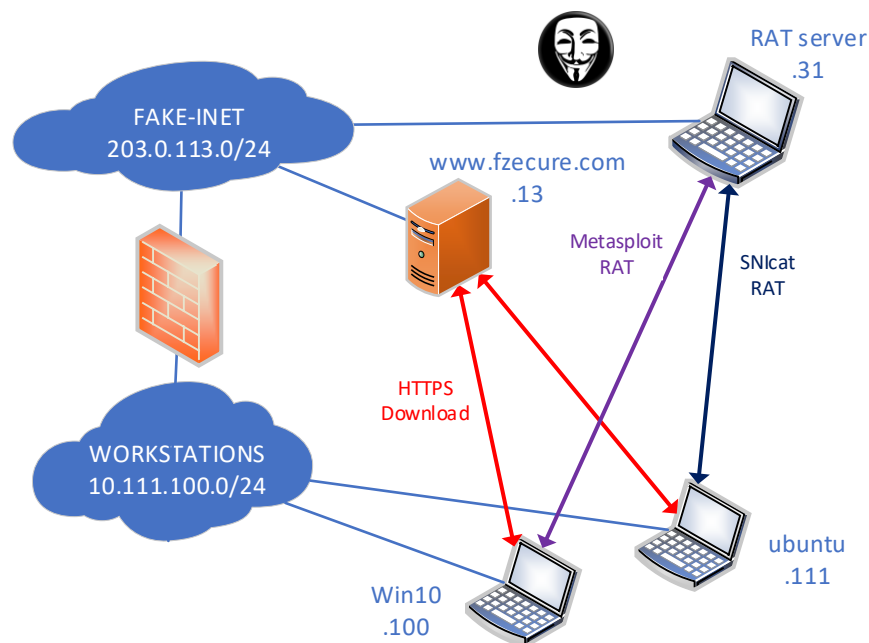


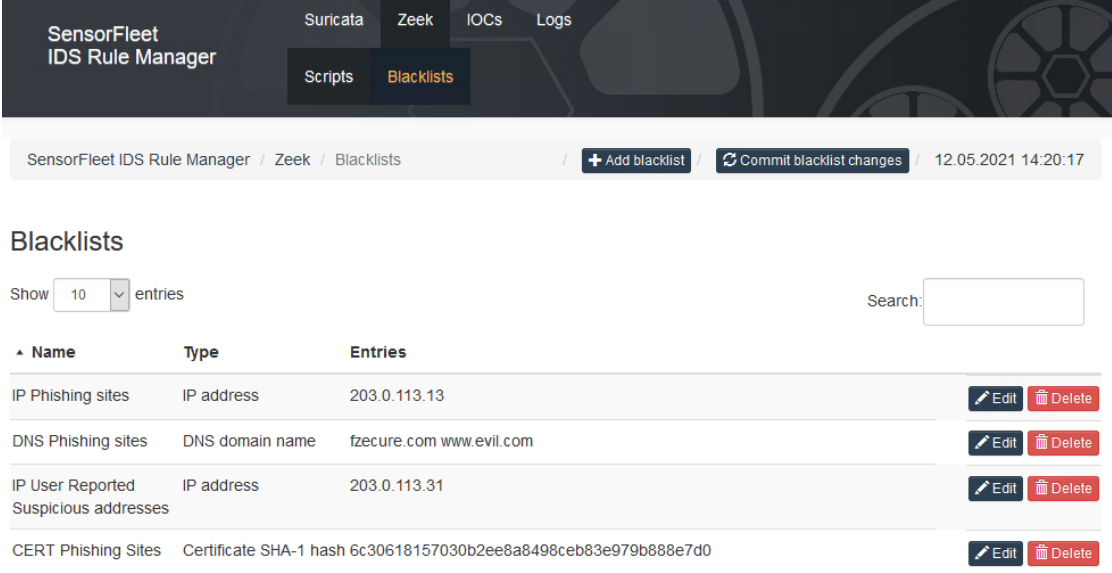
Figure 16. TLS traffic flows to phishing sites and RAT/C2 server

After attack simulations, more common detection cases often seen in real environments were tested. These tests included detecting Tor and DoH traffic and TLS misconfigurations in internal servers. Detection of port scanning, brute forcing and DDoS was left out since these attacks can be quite easily detected with traditional netflow based network monitoring tools. Detection of VPN traffic was also left out due to strict time schedule and since as a test case it wouldn't differ much from detecting Tor traffic.

6.2.1 Detecting Traffic to Phishing Site

At the very beginning of the attack simulation, workstations connect to phishing site www.fzecure.com and downloads RAT software. In this test scenario, phishing site is

already publicly known and blacklists for certificate hash, IP address and domain name are configured for ETA systems. Figure 17 displays blacklist configurations used for test scenario in SensorFleet IDS Rule Manager.



SensorFleet IDS Rule Manager / Zeek / Blacklists / + Add blacklist / Commit blacklist changes / 12.05.2021 14:20:17

Blacklists

Show 10 entries Search:

Name	Type	Entries	
IP Phishing sites	IP address	203.0.113.13	Edit Delete
DNS Phishing sites	DNS domain name	fzecure.com www.evil.com	Edit Delete
IP User Reported Suspicious addresses	IP address	203.0.113.31	Edit Delete
CERT Phishing Sites	Certificate SHA-1 hash	6c30618157030b2ee8a8498ceb83e979b888e7d0	Edit Delete

Figure 17. SensorFleet IDS Rule Manager Blacklists configuration

By investigating logs in LogPoint, matches for blacklist entries was seen. Figure 18 shows log entry forwarded by Sensorfleet Sensor indicating blacklisted certificate hash for phishing site was detected.

```
2021/05/12 14:55:02
log_ts=2021/05/12 14:55:02 | device_ip=10.222.0.82 | device_name=Sensorfleet | col_type=syslog | sig_id=7650000 | repo_name=default | severity=5 | facility=1 | @timestamp=2021-05-12T11:55:02.832Z | @version=1 | col_ts=2021/05/12 14:55:02 | collected_at=TestLogpoint | file_desc=203.0.113.13:443/tcp | file_mime_type=application/x-x509-user-cert | fluid=FKHyuF4rFAID2OvLBe | id-orig_h=10.111.100.100 | id-orig_p=50944 | id-resp_h=203.0.113.13 | id-resp_p=443 | logpoint_name=TestLogpoint | matched_0=Intel::CERT_HASH | norm_id=JSON | seen-indicator=6c30618157030b2ee8a8498ceb83e979b888e7d0 | seen-indicator_type=Intel::CERT_HASH | seen-node=zeek | seen-where=X509::IN_CERT | sources_0=CERT Phishing Sites | tags_0=sensorfleet_event | ts=1.620820501037431E9 | uid=CXaanA4wvRRzrsJ3Ri |
```

Figure 18. Blacklisted certificate hash detected in LogPoint

Figure 19 shows log entry which indicates match on domain blacklist created for phishing sites.

```

2021/05/12 14:55:02
log_ts=2021/05/12 14:55:02 | device_ip=10.222.0.82 | device_name=Sensorfleet | col_type=syslog | sig_id=7650000 | repo_name=default | severity=5 | facility=1 |
@timestamp=2021-05-12T11:55:02.823Z | @version=1 | col_ts=2021/05/12 14:55:02 | collected_at=TestLogpoint | fid=FL6xLU1jXKdMsleDAi | id-orig_h=10.111.100.100 |
id-orig_p=50945 | id-resp_h=203.0.113.13 | id-resp_p=443 | logpoint_name=TestLogpoint | matched_0=Intel::DOMAIN | norm_id=JSON | seen-indicator=fzecure.com |
seen-indicator_type=Intel::DOMAIN | seen-node=zeek | seen-where=X509::IN_CERT | sources_0=DNS Phishing sites | tags_0=sensorfleet_event | ts=1.620820500738402E9
| uid=CHE9SGlwTNG1wvDf |

```

Figure 19. Blacklisted domain name detect in LogPoint

These logs are generated by Zeek at SensorFleet Sensor by analyzing x509.logs against Zeek's threat intel framework, which gets its blacklists from SensorFleet Manager. Both logs shows that connection was made from IP used by windows workstation (10.111.100.100) to IP assigned for www.fzecure.com (203.0.113.13). Figure 20 show log entry indicating match for phishing sites IP blacklists. This log is generated by analyzing Zeek's conn.logs and matching against IP blacklist in threat intel framework.

```

2021/05/12 14:55:02
log_ts=2021/05/12 14:55:02 | device_ip=10.222.0.82 | device_name=Sensorfleet | col_type=syslog | sig_id=7650000 | repo_name=default | severity=5 | facility=1 |
@timestamp=2021-05-12T11:55:02.819Z | @version=1 | col_ts=2021/05/12 14:55:02 | collected_at=TestLogpoint | id-orig_h=10.111.100.100 | id-orig_p=50945 |
id-resp_h=203.0.113.13 | id-resp_p=443 | logpoint_name=TestLogpoint | matched_0=Intel::ADDR | norm_id=JSON | seen-indicator=203.0.113.13 |
seen-indicator_type=Intel::ADDR | seen-node=zeek | seen-where=Conn::IN_RESP | sources_0=IP Phishing sites | tags_0=sensorfleet_event | ts=1.620820500738329E9 |
uid=CHE9SGlwTNG1wvDf |

```

Figure 20. Blacklisted IP address detected in LogPoint

Security Onion wasn't configured to use any static blacklists. However, by using Security Onion's "Hunt" functionality, similar Zeek logs leading to same threat information was found by using keywords *x509.certificate.subject*, *x509.certificate.serial* and *destination.ip*.

6.2.2 Detecting Metasploit HTTPS Reverse Shell traffic

After downloading Metasploit RAT client file from phishing site to windows workstation, the malicious exe file was run to initiate TLS connection from workstation to Metasploit RAT server running at attacker's Kali VM in internet. Metasploit's reverse HTTPS remote shell functionality was used with its self-signed certificate. Value of the certificate subject field varies by instances to make static matching against blacklists hard but by using self-signed certificate for service at internet would make it stand out from regular HTTPS traffic. In testing scenario,

network traffic coming from IP address of the Metasploit server 203.0.113.31 has been categorized to be suspicious by company's external threat intel partner and IP was configured on SensorFleet IP blacklist as seen in Figure 17. Since Metasploit remote shells are commonly used and seen on network, it is very likely it could be detected by matching JA3 and JA3s hashes. Suricata detection rules used by both ETA systems was implemented using dynamic Suricata rules from community supported free to use website rules.emergingthreats.net. Static rules matching for JA3 and JA3s hashes wasn't implemented in this scenario.

After Metasploit remote shell was established, first indication was discovered at Security Onion's alert portal. There was alert for JA3 hash possible match for Trickbot malware as seen on Figure 21.

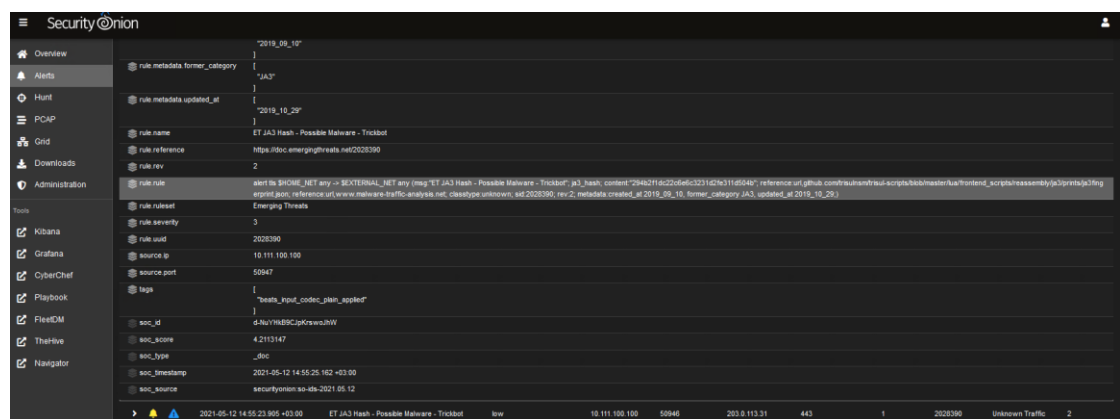


Figure 21. Security Onion alert for JA3 match for malware

Alert details *rule.rule* field shows the exact Suricata rule with JA3 hash value, which generated the alert. As destination IP on the alert matched with suspicious IP 203.0.113.31, further investigating was made in Security Onions Hunt portal using filter with JA3 hash which was seen on alert (294b2f1dc22c6e6c3231d2fe311d504b). This resulted information from Zeek logs related to event which triggered the alert as seen on Figure 22.

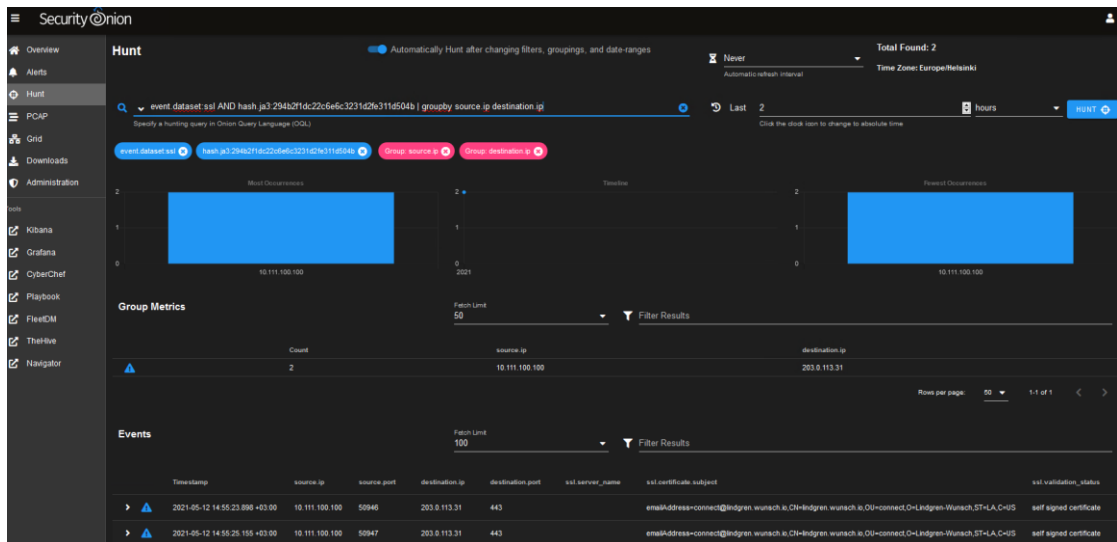


Figure 22. Investigating alert in Security Onion's Hunt portal

This log entry stated that self-signed certificate was used and order of attributes in `ssl.certificate.subject` field seemed abnormal (emailAddress,CN,OU,O,ST,C). These can both be considered IoCs. JA3 and JA3s hashes were further investigated using publicly available API for community supported JA3 hash database site ja3er.com. JA3 hash resulted match for Trickbot but JA3s hash didn't have a match as seen on Figure 23.

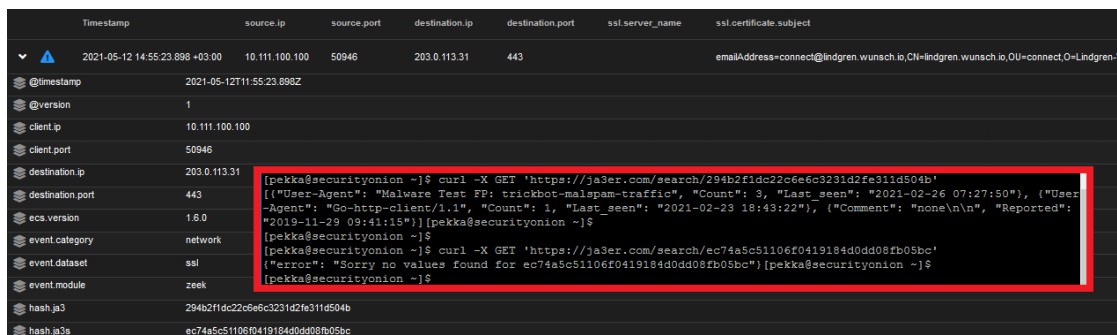


Figure 23. Querying JA3 hashes from ja3er.com database

It seemed odd at first why JA3 hash of Metasploit TLS connection would match famous banking trojan Trickbot but further investigations revealed that Trickbot actually utilize Metasploit among other frameworks (Dahan, Rochberger, Salem, Zhao, Yona, Yampel, Hart 2019).

Connection to Metasploit server's IP address was also identified in LogPoint using IP blacklist in similar fashion as connection to phishing site. By looking log entry details, same IoCs which was seen in SecurityOnion was seen in LogPoint as well as seen on Figure 24. Only differ is that log entry in LogPoint doesn't directly state that self-signed certificate is being used but it shows that certificate subject and issuer are the same.

```
2021/05/12 14:55:28
log_ts=2021/05/12 14:55:28 | device_ip=10.222.0.82 | device_name=Sensorfleet | col_type=syslog | sig_id=7650000 | repo_name=default | severity=5 | facility=1 |
subject=emailAddress=connect@lindgr... | @timestamp=2021-05-12T11:55:28.549Z | @version=1 | cert_chain_fuids_0=FY9Dsb4H3yt4GKySc |
cipher=TLS_ECDHE_RSA_WITH_AES_256_... | col_ts=2021/05/12 14:55:28 | collected_at=TestiLogpoint | curve=x25519 | established=true | id-orig_h=10.111.100.100 |
id-orig_p=50947 | id-resp_h=203.0.113.31 | id-resp_p=443 | issuer=emailAddress=connect@lindgr... | ja3=294b2f1dc22c6e6c3231d2fe311... |
ja3s=ec74a5c51106f0419184d0dd08f... | logpoint_name=TestiLogpoint | norm_id=JSON | tags_0=sensorfleet_event | ts=1.620820525163448E9 | uid=CsuYiZ3aq0Y3TidZd9
| version=TLSv12
```

Figure 24. Log entry for Metasploit HTTPS remote shell in LogPoint

After verifying that ETA systems have detected Metasploit HTTPs remote shell connection, data theft was initiated by using Metasploit meterpreter's inbuilt download functionality. Large file was transferred from workstation to Kali VM over HTTPS remote shell. This should be seen as abnormal traffic behavior since workstations generally doesn't generate much upstream traffic to internet. Remote shell session was terminated after one and half hour to generate a TLS connection log that could also be detected by longer than average duration. By using search parameters in LogPoint looking for TLS sessions with upstream more 100MB or duration more than one hour, connection used for data theft stand out as seen on Figure 25.

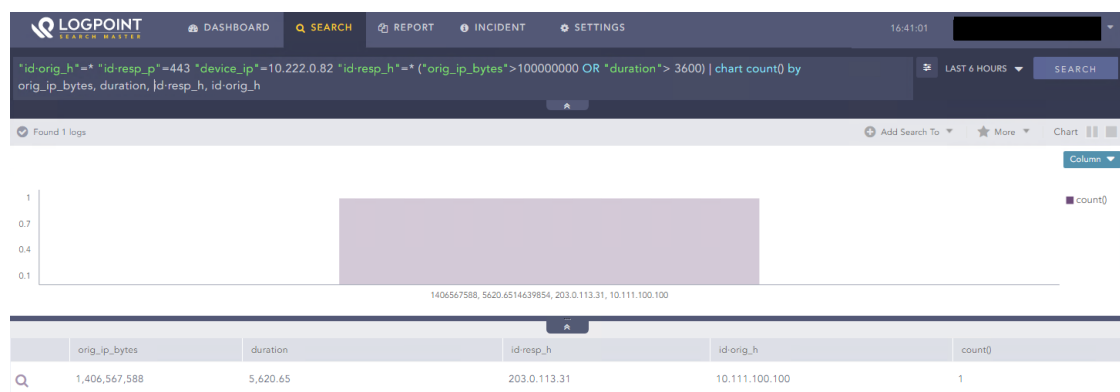


Figure 25. TLS connection in LogPoint indicating possible data theft

Similar query was done in Security Onion and similar log entry generated from Zeek's conn.logs was identified. Security Onion's Zeek logs of testing day was copied and uploaded to database in RITA server for further analysis. RITA detected beaconing activity from windows workstation to Kali VM with risk score of 0,627 as seen on Figure 26. It also detected connections from workstation to OpenDNS servers as high-risk beaconing activity. This is because RITA was not configured to exclude these known good IP addresses from beaconing analysis. RITA also detected the long duration of TLS sessions between workstation and Kali in its "Long Connections" analysis.

Score	Source	Destination	Connections	Avg. Bytes	Intvl. Range	Size Range	Intvl. Mode	Size Mode	Intvl. Mode Count	Size Mode Count	Intvl. Skew	Size Skew	Intvl. Dispersion	Size Dispersion	Total Bytes
0.789	10.111.100.100	208.67.222.222	6533	250.000	13282	325	4	228	1992	1995	0.333	0.000	1	21	1636394
0.723	10.111.100.100	208.67.222.220	6529	258.000	13283	310	4	304	2010	1995	0.333	-0.053	1	36	1687790
0.627	10.111.100.100	203.0.113.31	134	5270219.000	16192	703990055	5	729	25	28	0.000	-0.177	2	163	706209388

Source	Destination	DstPort:Protocol:Service	Duration
10.111.100.100	203.0.113.31	0:icmp-, 443:tcp-, 443:tcp:ssl	5620.4965

Figure 26. IoCs in RITA for Metasploit HTTPS remote shell

TLS connections with large upstream and long durations are not uncommon in certain environments and specifying threshold values for these types of connections to be considered as IoCs can be very tricky. Known good IP addressed should be whitelisted when implementing alerting.

6.2.3 Detecting SNIcat C2 traffic

SNIcat is a python-based Proof of Concept (PoC) tool used to demonstrate how C2 data can be tunneled using TLS SNI field. This is a covert channel which bypassed many firewall and proxies in the beginning of 2020, when the tool was developed. SNIcat server utilizes both CA signed wildcard certificates and self-signed certificates to tunnel traffic from client to server. Kali VM used as SNIcat server was configured to use wildcard certificate **.fzecure.com* signed by testing environments CA and self-

signed certificate with subject *update.fzecure.com* to make it seem like a legit update server for antivirus client.

SNlicat client software was downloaded to ubuntu workstation from same phishing site *www.fzecure.com*. Detection of this traffic in ETA systems was very similar to one in previous chapter and is therefore not described further. When starting SNlicat client python program, it is given parameters for server IP, TCP port and certificates. It will then try repeatedly to communicate with specified server. After server is started, it can see connected clients and send basic filesystems commands to them.

This time Security Onion was configured with static Suricata rule which would detect the **.fzecure.com* wildcard certificate and it did raise an alert as seen on Figure 27.

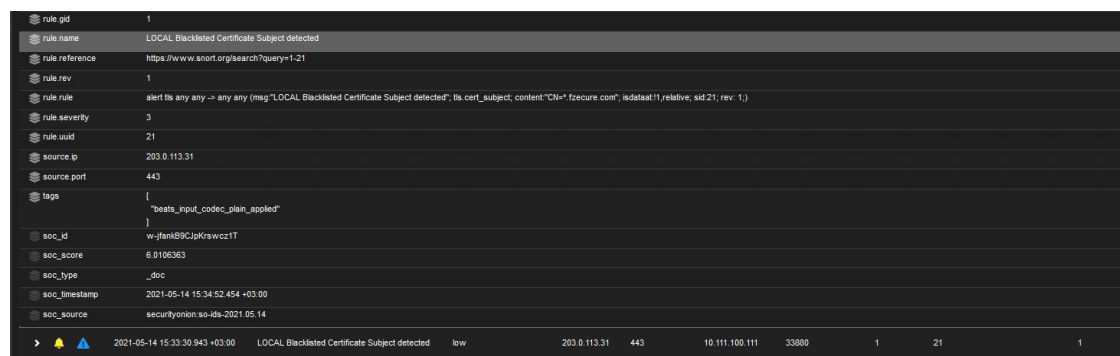


Figure 27. Security Onion Suricata alert for blacklisted certificate subject

When logs were investigated using keyword *ssl.server_name* in Security Onion Hunt portal, it showed that data in SNI field was in format *<cmd>-<data>.fzecure.com*, where *cmd* is SNlicat command sent to client and *data* is data bytes tunneled in SNI field. Figure 28 displays examples of commands and data in SNI fields.

Timestamp	source_ip	source_port	destination_ip	destination_port	ssl_server_name	ssl_certificate.subject	ssl_validation_status
2021-05-14 15:34:52.546 +03:00	10.111.100.111	34208	203.0.113.31	443	fmito-P9GwasrQlniVjAIPupdate.fzecure.com	CN=*.fzecure.com,O=F-Zecure,L=Helsinki,ST=Some-State,C=FI	unable to get local issuer certificate
2021-05-14 15:35:21.488 +03:00	10.111.100.111	34324	203.0.113.31	443	LIST-dofF4yoY9bzVJraupdate.fzecure.com	CN=*.fzecure.com,O=F-Zecure,L=Helsinki,ST=Some-State,C=FI	unable to get local issuer certificate
2021-05-14 15:34:49.108 +03:00	10.111.100.111	34192	203.0.113.31	443	LIST-rTtBx5HbX7VvaATupdate.fzecure.com		self signed certificate
2021-05-14 15:34:49.655 +03:00	10.111.100.111	34194	203.0.113.31	443	SIZE-VyYpJpDqRRaQZMAOupdate.fzecure.com		self signed certificate
2021-05-14 15:35:20.359 +03:00	10.111.100.111	34205	203.0.113.31	443	EX-70iBlnIZdnHY6axJupdate.fzecure.com		self signed certificate
2021-05-14 15:34:52.407 +03:00	10.111.100.111	34204	203.0.113.31	443	LS-0iBtucp6IDFv8update.fzecure.com	CN=*.fzecure.com,O=F-Zecure,L=Helsinki,ST=Some-State,C=FI	unable to get local issuer certificate
2021-05-14 15:33:30.330 +03:00	10.111.100.111	33878	203.0.113.31	443	EX-y6fNGoqNtL9SrL7update.fzecure.com		self signed certificate
2021-05-14 15:05:41.650 +03:00	10.111.100.111	33654	203.0.113.31	443	ALIVE-uHkMkmcYoy94Kfzecure.com	CN=update.fzecure.com,O=F-Zecure,L=Helsinki,ST=Some-State,C=FI	unable to get local issuer certificate
2021-05-14 14:54:52.878 +03:00	10.111.100.111	60160	203.0.113.31	443	CB-yyEzOaQc5ebOAYHfzecure.com		self signed certificate
2021-05-14 14:54:51.247 +03:00	10.111.100.111	60154	203.0.113.31	443	LD-QD3PHpDwWSSWNDorfzecure.com		self signed certificate

Figure 28. Detecting SNlcat C2 traffic in Security Onion

Same kind of information was also available in LogPoint when querying for Kali virtual machine IP and looking for *server_name* and *subject* keywords as seen on Figure 29.

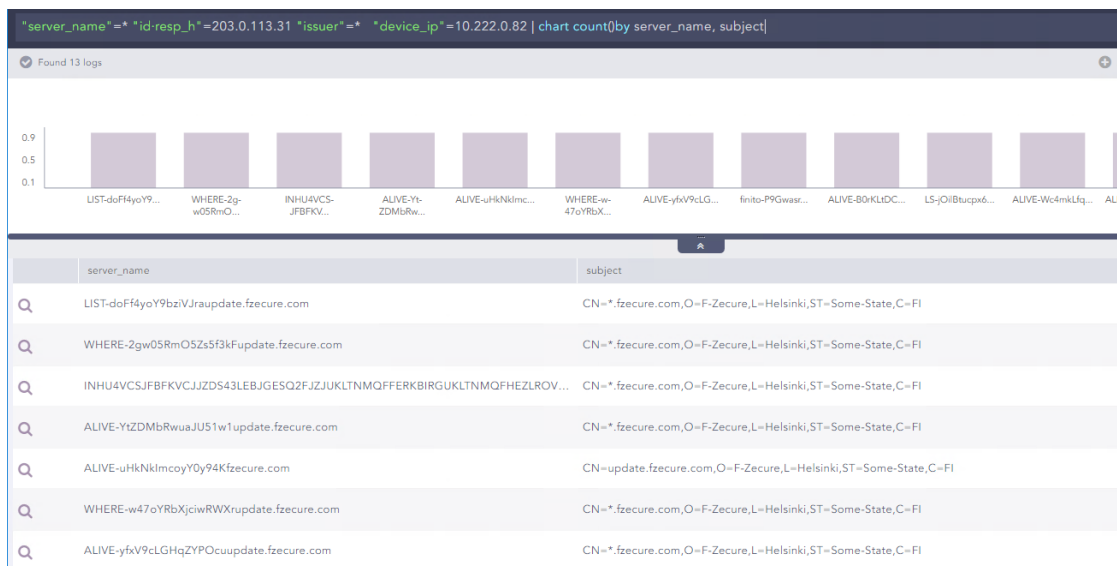


Figure 29. Detecting SNlcat C2 traffic in LogPoint

JA3 and JA3s hashes was seen and noticed to be identical for every connection in SNlcat. JA3 hash for was 6f16291393bca9be2dd25cc7ad01f971 and JA3s hash was c74a5c51106f0419184d0dd08fb05bc. Both hashes were queried using ja3er.com API but they didn't provide any relevant information as JA3 hash matched with *"Mozilla/5.0 (Windows NT 5.1; U; en; rv:1.8.1) Gecko/20061208 Firefox/2.0.0 Opera*

9.51” and JA3s didn’t have a match. This is probably because SNIcat is only a PoC tool suitable for testing and not being actively used by threat actors.

Once again, Zeek logs from testing time was copied and uploaded from Security Onion to database in RITA server and analyzed. RITA identified SNIcat beacons with top score as seen on Figure 30.

Score	Source	Destination	Connections	Avg. Bytes	Intvl. Range	Size Range	Intvl. Mode	Size Mode	Intvl. Mode Count	Size Mode Count	Intvl. Skew	Size Skew	Intvl. Dispersion	Size Dispersion	Total Bytes
1.000	10.111.100.111	203.0.113.31	11229	120.000	2596	994	1	60	5642	11114	0.000	0.000	0	0	1349306
0.666	10.111.100.100	208.67.222.222	3498	248.000	129	251	4	228	1147	1087	0.667	0.486	1	18	869420
0.664	10.111.100.100	208.67.222.220	3465	263.000	49	264	4	304	1151	1087	0.600	-0.368	1	24	912882
0.275	10.111.100.111	8.8.8.8	591	1681.000	599	5481	300	2064	79	232	0.888	-0.875	28	24	993548

Figure 30. Detecting SNIcat beaconing in RITA

SNIcat connection wasn’t working reliably in testing environment. Therefore only basic C2 functionality was tested and file transfer over SNIcat was left outside of testing.

6.2.4 Detecting Tor

ETA systems capabilities to detect real Tor traffic was tested by using Tor browser in ubuntu workstation and doing some browsing via Tor network. First, Tor circuit was investigated using Tor browser’s Site Information view and browsed to ripe.net to verify that connection to internet is coming from Tor network instead of test environments internet connection. In this case, connection to Tor is done using Tor node in United Kingdom with IP 77.68.88.20 as seen on Figure 31.

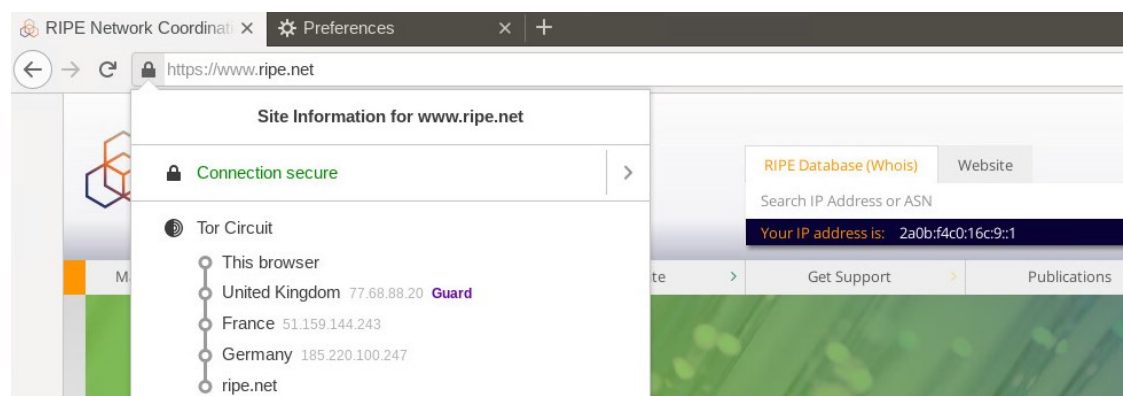


Figure 31. Investigating Tor Circuit and external IP in Tor browser

Suricata in Security Onion identified connection to known Tor IP address and created alert seen on Figure 32. Similar Suricata alert was also seen in LogPoint, which wasn't surprising since both ETA systems use dynamic ET rules which have up to date information on IP addresses used by Tor exit nodes.

rule.name	ET TOR Known Tor Relay/Router (Not Exit) Node Traffic group 709
rule.reference	https://doc.emergingthreats.net/2522708
rule.rev	4430
rule.rule	alert tcp [77.6.185.95,77.68.122.236,77.68.126.56,77.68.30.104,77.68.3.252,77.68.79.72,77.68.7.99,77.68.88.20,77.68.88.55,77.68.94.106] any -> \$HOME_NET any (msg:"ET TOR Known Tor Relay/Router (Not Exit) Node Traffic group 709" threshold: type limit, track by_src, seconds 60, count 1, classtype:misc-attack, flowbits:set,ET:TorIP, sid:2522708, rev:4430, metadata:affected_product Any, attack_type, created_at 2008_12_01, updated_at 2021_05_14.)
rule.ruleset	Emerging Threats
rule.severity	2
rule.iuid	2522708
source.geo.continent_name	Europe
source.geo.country_iso_code	GB
source.geo.country_name	United Kingdom

Figure 32. Security Onion alert for known Tor node

By filtering logs in Security Onion with Tor node IP address, Zeek's ssl.logs provided JA3 and SNI IoCs for Tor traffic as seen on Figure 33. JA3 hash for Tor traffic was *711528629b81edc0307f28392d2a96c0*, JA3s hash was *15af977ce25de452b96affa2addb1036* and SNI for this connection was *www.uemlnbynvd3qyfmum4nm4b6k.com*, which seems random and stands out from SNIs used by regular web sites.

destination_geo.country_name	United Kingdom
destination_geo.ip	77.68.88.20
destination_geo.location.lat	51.4964
destination_geo.location.lon	-0.1224
destination_geo.timezone	Europe/London
destination.ip	77.68.88.20
destination.port	443
ecs.version	1.6.0
event.category	network
event.dataset	ssl
event.module	zeek
hash.ja3	711528629b81edc0307f28392d2a96c0
hash.ja3s	15af977ce25de452b96affa2addb1036
ingest.timestamp	2021-05-15T08:09:22.857Z
log.file.path	/hsm/zeek/logs/current/ssl.log
log.id.uid	CUQs0B2h9HSb7nqD3g
log.offset	9712
message	[{"ts":"2021-05-15T08:09:16.952353Z","uid":"CUQs0B2h9HSb7nqD3g","id_orig_h":"10.111.100.111","id_orig_p":58470,"id_resp_h":"77.68.88.20","id_resp_p":443,"version":"TLSv13","cipher":"TLS_AES_256_GCM_SHA384","sni":"www.uemlnbynvd3qyfmum4nm4b6k.com","resumed":false,"established":true,"ja3":"711528629b81edc0307f28392d2a96c0","ja3s":"15af977ce25de452b96affa2addb1036"}]
observer.name	securityonion
server.ip	77.68.88.20
server.port	443
source.ip	10.111.100.111
source.port	58470
ssl.cipher	TLS_AES_256_GCM_SHA384
ssl.curve	secp256r1
ssl.established	true
ssl.resumed	false
ssl.server_name	www.uemlnbynvd3qyfmum4nm4b6k.com
ssl.version	TLSv13

Figure 33. IoCs for Tor traffic seen in Security Onion

Same IoCs were seen on LogPoint. By using Tor browser’s JA3 hash as a filter and querying and creating Sankey chart for source and destination IPs, SNI and JA3 and JA3s hashes visualization seen on Figure 34 indicates that while JA3 and JA3s hashes remains the same for many connections, entry nodes change from time to time and SNI used by Tor connection seems to change quite often.



Figure 34. LogPoint Sankey chart for Tor traffic

Using ja3er.com API for Tor JA3 hash resulted match: `{"User-Agent": "Tor Browser 9.0.2", "Count": 1, "Last_seen": "2020-02-02 19:23:10"}`. There was no match for JA3s hash. In RITA, Tor connection to entry node was seen listed in “Long Connections” portal indicating unnormal TLS activity.

6.2.5 Detecting DoH

DoH detection was tested by configuring ubuntu workstation’s Firefox to use DoH and Cloudflare as a service provider and doing web browsing to generate traffic.

Suricata in Security Onion alerted for DoH by detecting “*cloudflare-dns.com*” in SNI field as seen in Figure 35.

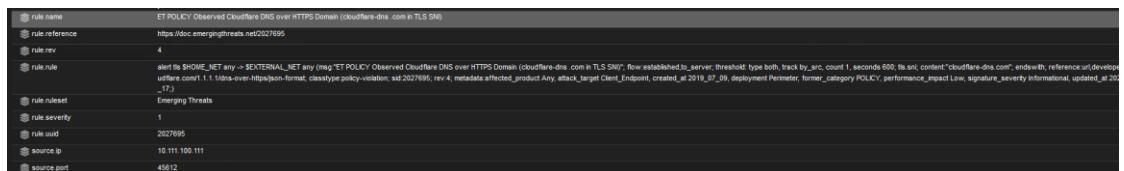


Figure 35. Security Onion Suricata alert for DoH to cloudflare.com

Suricata alert showed that destination IP address was 104.16.249.249. This IP was used for filtering Zeek’s ssl.logs to get more IoCs such as exact SNI being *mozilla.cloudflare-dns.com* as seen on Figure 36. Similar metadata information for DoH was seen also in LogPoint.

Timestamp	source.ip	source.port	destination.ip	destination.port	ssl_server_name	ssl_certificate.subject	ssl_validation_status	ssl.version
2021-05-15 10:38:24.016 +03:00	10.111.100.111	45612	104.16.249.249	443	mozilla.cloudflare-dns.com			TLSv13
@timestamp	2021-05-15T07:38:24.016Z							
@version	1							
client.ip	10.111.100.111							
client.port	45612							
destination.geo.continent_name	North America							
destination.geo.country_iso_code	US							
destination.geo.country_name	United States							
destination.geo.ip	104.16.249.249							
destination.geo.location.lat	37.751							
destination.geo.location.lon	-97.822							
destination.geo.timezone	America/Chicago							
destination.ip	104.16.249.249							
destination.port	443							
ecs.version	1.6.0							
event.category	network							
event.dataset	ssl							
event.module	zeek							
hash.ja3	aa7744226c695c0b2e440419848cf700							
hash.ja3s	eb1d94daa7e0344597e756a1fb6e7054							
ingest.timestamp	2021-05-15T07:38:32.756Z							
log.file.path	/nsm/zeek/logs/current/ssl.log							

Figure 36. IoCs for Cloudflare DoH traffic seen on Security Onion

JA3 hash *aa7744226c695c0b2e440419848cf700* queried from ja3er.com resulted match for Firefox browser. It first seemed that this was generic hash for Firefox browser which would match also normal browsing traffic but when filtering logs using this hash it resulted only connection to Cloudflare DoH. Therefore, this hash can be used as IoC for detecting DoH for this specific client but is not suitable for detecting DoH for other clients. JA3s didn't result any match from ja3er.com. In RITA, DoH connection was seen listed in "Long Connections" portal indicating unnormal TLS activity.

6.2.6 Detecting TLS Misconfigurations

ETA systems can also be used to detect TLS misconfigurations and policy violations such as usage of legacy and vulnerable TLS version or ciphersuites and self-signed or invalid x509 certifications inside organization network. In testing environment company Lupari Oy had two servers running TLS on network 10.111.1.0/24. One of the servers was configured to use self-signed certificate and another was configured to use certificate signed by company's own internal CA, but it was configured to support only legacy TLS versions and ciphersuites. First, HTTPS traffic was generated to servers from company's workstations and then ETA systems were used to query and visualize information regarding company's TLS and certification metadata. Using query seen on Figure 37, LogPoint presented information indicating that server

db.int.lupari.fi supports acceptable TLS version and ciphersuite but has self-signed certificate as certificate issuer and subject are the same. Server intra.int.lupari.fi on the other hand was seen to have certificate signed by company's internal CA but is not supporting recommended TLS versions and ciphersuites.

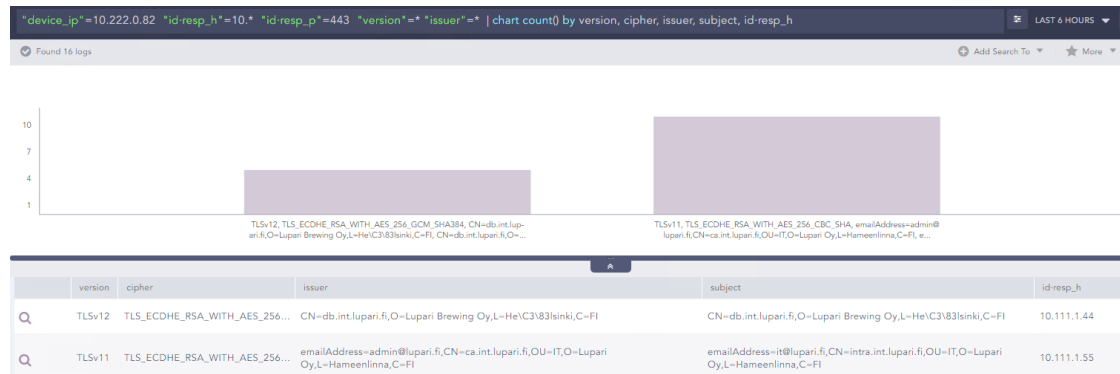


Figure 37. Detecting TLS misconfigurations in LogPoint

Similar information was seen with two queries used in Security Onion. First query seen on Figure 38, presents TLS versions and ciphersuites for servers in lupari.fi domain.

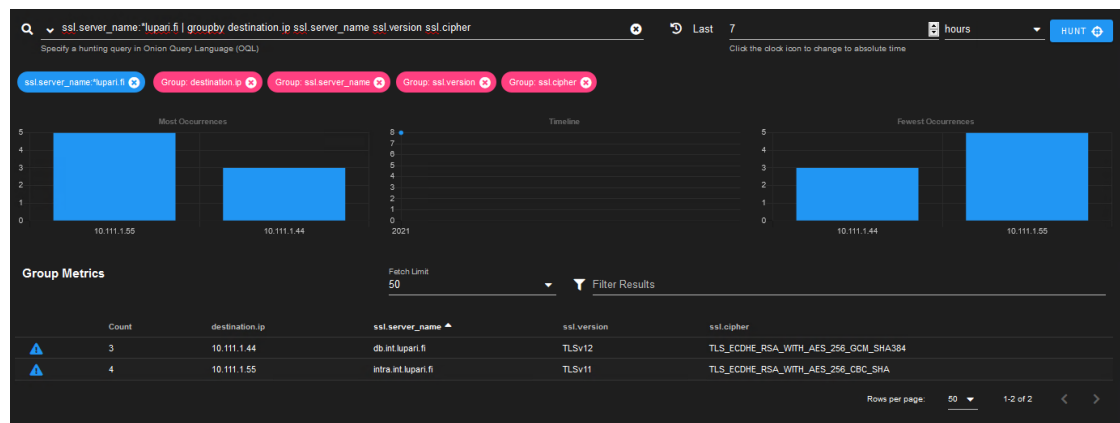


Figure 38. Detecting TLS version and ciphersuites in Security Onion

Second query, as seen on Figure 39, presents certificate information for same servers. Security Onion has additional ssl.validation_status keyword which make it even easier to detect self-signed certificates.

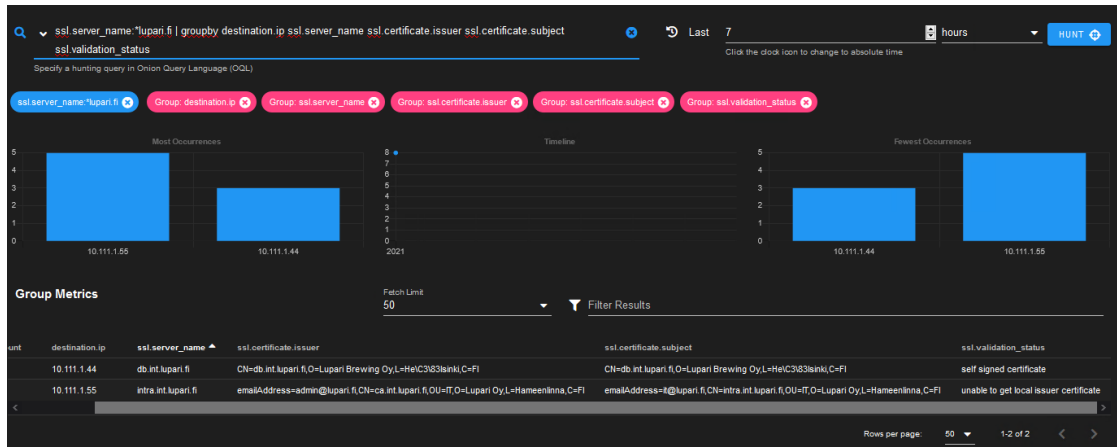


Figure 39. Detecting x.509 misconfigurations in Security Onion

6.3 Administrative and Operational Evaluation

Both ETA systems performed reliably and smoothly. Query syntax was similar in LogPoint and Security Onion. LogPoint is more versatile with visualization but Security Onion on the other hand has many inbuilt functionalities for parsing Zeek logs, such as indicating directly self-signed certificates. From security analyst perspective, both systems were easy to use and finding relevant information using queries was fast if one already knows what to look for.

From the administrative perspective, ETA system one was heavier to implement than ETA system two and required quite a lot of work to get it initially running. Installation and configuration of Sensorfleet components wasn't straightforward but scaling it with additional sensors should be much easier since most of the settings can be pushed to new sensors from the manager server. Installing LogPoint SIEM was out of the scope of this thesis as it was already present in the testing environment.

ETA system two was surprisingly straightforward to setup with default configuration. Both Security Onion and RITA required minimal administrative effort to get them running at a decent level. There weren't many additional configurations done for these systems. Security Onion was configured with few static Suricata rules, but RITA was run with default configuration for all the tests. Security Onion was installed in single

node mode but if it would be implemented with distributed architecture it would probably be much time consuming to implement.

7 Research Results

Research results includes results from testing phase and results to research objectives. First test results are analyzed and then results to objectives is being evaluated.

7.1 Test Results

In testing phase, ETA solutions weren't benchmarked against each other in detail. They were however compared in high level by evaluating easiness of implementation, administration and operation and how ETA systems were able to detect and visualize IoCs in testing scenarios. Comparison was done using scoring with scale from one to three. When scoring easiness of implementing ETA components, value one indicates that implementation required more than average amount of effort and time before system was up and running with default settings. Value three indicate that implementation was easier than average. Same logic was used when scoring easiness of general administrative and operational tasks of the systems. LogPoint SIEM was only scored from operational perspective since it was already implemented in Cinia's testing environment. Therefore it wasn't possible to do strictly mathematical comparison between ETA solutions but one could summarize that ETA1 was harder to implement than ETA2 but when comparing administrative and operational functionalities there wasn't much difference. Scores for implementation, administration and operation of ETA solutions can be seen in Table 7.

Table 7. Ease of implementation, administration and operation of ETA solutions

ETA System	ETA1		ETA2	
Subsystem	SensorFleet	LogPoint	Security Onion	RITA
Implementation	1	-	3	3
Administration	2	-	2	3
Operation	3	3	3	3

When comparing capabilities of ETA systems to detect IoCs, simple binary scoring logic was used, where zero means IoC wasn't detected and one means it was detected. Pyramid of pain layers related to IoCs are listed in tables as well.

Both ETA systems were capable to detect TLS connections destined to known bad web site using IoCs for certificate hash, destination IP address and domain name seen on certificate subject field. Scores for detecting IoCs for phishing site used in tests are seen on Table 8.

Table 8. Detecting IoCs for Phishing Site

IoC Type	Pyramid of Pain	ETA1	ETA2
Certificate Hash	Hash Values	1	1
Destination IP	IP address	1	1
Certificate Subject	Domain Names	1	1
Score		3	3

Both ETA systems were able to detect usage of self-signed certificate, however there was minor difference how ETA systems displays it. Security Onion Hunt portal in ETA2 indicates it directly using keyword *ssl.validation_status* in LogPoint same result can be seen by comparing certificate subject to certificate issuer. Similar mechanism as used in ETA2 could be added to ETA1 with minor effort directly into Sensorfleet using Zeek's scripting or with enrichment configuration in LogPoint but it wasn't done due to strict schedule used for testing phase of the research. IoCs for

destination IP and certification subjects were seen similarly as for phishing site. JA3 and JA3s hashes were seen indicating JA3 match for Trickbot, which wasn't the actual malware used in testing but would be enough for any security analyst to start further investigations. Beaconing was detected by RITA in ETA2. Beaconing detection in ETA1 wasn't easy during testing but might be possible to do if all zeek log fields would be normalized and analyzed by LogPoints UEBA module. On the other hand, one solution would be to use additional RITA server for ETA1 and forward zeek logs also there for beaconing analysis. Since RITA is opensource product there wouldn't be much additional costs. IoCs for exceptional upload traffic and long connection was detected by both ETA systems. Scores for detecting IoCs for Metasploit HTTPS reverse shell and data theft done in tests are seen on Table 9.

Table 9. Detecting IoCs for Metasploit HTTPS reverse shell and data theft

IoC Type	Pyramid of Pain	ETA1	ETA2
Self-signed Certificate	Hash Values	1	1
Destination IP	IP address	1	1
Certificate Subject	Domain Names	1	1
JA3/JA3s Hashes	Network Artifacts/Tools	1	1
Beaconing	Network Artifacts	0	1
Long Connections	Network Artifacts	1	1
Unusual Upstream Traffic Rates	Network Artifacts	1	1
Score		6	7

When testing SNIcat C2 traffic, both ETA systems detected IoCs for self-signed certificate, known blacklisted IP address and SNIs containing C2 data for blacklisted domain. JA3 and JA3s hashes of SNIcat were also detected but these hashes doesn't seem to exist in public JA3/JA3s databases. IoC for beaconing activity was detected by ETA2 but not ETA1 for reasons discribed earlier. Scores for detecting IoCs for SNIcat C2 traffic generated in tests are seen on Table 10.

Table 10. Detecting IoCs for SNIcat C2 traffic

IoC Type	Pyramid of Pain	ETA1	ETA2
Self-signed Certificate	Hash Values	1	1
Destination IP	IP address	1	1
SNI	Domain Names	1	1
JA3/JA3s Hashes	Network Artifacts/Tools	1	1
Beaconing	Network Artifacts	0	1
Score		4	5

Both ETA systems used dynamic Suricata rules from Emerging Threats, which had comprehensive listings of known Tor edge nodes. Therefore IoC for IP address was easily detected by both systems. IoCs for SNI containing suspiciously looking strings, JA3 hash matching Tor browser and long connection between Tor client and entry node were detected by both ETA systems. Eventhough not used as scoring criteria, sankey visualization in LogPoint added additional value when putting all the IoCs together. Scores for detecting IoCs for Tor traffic generated in tests are seen on Table 11.

Table 11. Detecting IoCs for Tor traffic

IoC Type	Pyramid of Pain	ETA1	ETA2
Destination IP	IP address	1	1
SNI	Domain Names	1	1
JA3/JA3s Hashes	Network Artifacts/Tools	1	1
Long Connections	Network Artifacts	1	1
Score		4	4

DoH was detected mainly by IoC for SNI containing domain name for known DoH service provider. Additional IoCs included destination IP belonging to known DoH

service provider and long connection between DoH client and server. Both ETA systems detected these IoCs. JA3 and JA3s hashes were also detected but these are weak IoCs since the range of possible hashes for DoH is huge. If DoH is configured to use uncommon DoH servers, for example DoH servers managed by threat actor, detection of DoH would become difficult. Scores for detecting IoCs for DoH traffic generated in tests are seen on Table 12.

Table 12. Detecting IoCs for DoH traffic

IoC Type	Pyramid of Pain	ETA1	ETA2
Destination IP	IP address	1	1
SNI	Domain Names	1	1
Long Connections	Network Artifacts	1	1
Score		3	3

Detecting and visualizing TLS misconfiguration and policy violation was easily done in both ETA systems. Scores for detecting TLS misconfigurations and policy violations in testing environment are seen on Table 13.

Table 13. Detecting IoCs for TLS misconfigurations and policy violations

IoC Type	Pyramid of Pain	ETA1	ETA2
Self-signed certificate	Hash Values	1	1
TLS version	Network Artifacts	1	1
Ciphersuites	Network Artifacts	1	1
Score		3	3

Overall, both ETA systems performed well and there wasn't many significant differences between core functionalities.

7.2 Evaluation of Research Objectives

Primary objective in research was to gain understanding of methods and tools how one can detect anomalies in TLS encrypted traffic without decrypting it and how opensource products could be utilized. Subject was relatively new to researcher before starting thesis. During thesis process, understanding what ETA means and what frameworks, methods and tools are available to address the subject, become clear. It also become clear that these methods, tools and systems are constantly changing. One ETA system, Apache Metron, which researcher investigated and initially planned to test in practice was retired by 2021 and was therefore left out of the thesis. However, some of the tools, such as Suricata and Zeek, are long lasting and have been used widely for many years. Frameworks, such as Pyramid of Pain, are also long lasting and can be used as guidelines when designing and implementing ETA solutions. In researcher's opinion, basic methods, frameworks and tools used for ETA was explained and understanding was gained. It also became clear that there are many opensource products available to be utilized for ETA. Most of them are good for certain specific task but there aren't many products that would provide full-scale ETA solution on their own. Security Onion comes close to that but it lacks ML algorithms for behavior-based detection which most commercial ETA solutions offer.

Second objective was to investigate ETA capabilities of combination of SensorFleet and LogPoint. Investigations at this point reveals that they can be utilized together to perform ETA roughly at the same level as one could do with combination of Security Onion and RITA. Utilizing LogPoint's advanced features such as UEBA for ETA however hasn't been tested yet and this testing will probably continue afterwards in Cinia.

8 Conclusions

Based on test results, opensource product can be used for ETA, but one should consider use cases and environments where they are suitable. For example, gathering and storing metadata of TLS traffic can be easily done with opensource tools, such as Zeek, and that can be enough for many organizations. This would allow organization to investigate TLS traffic on metadata level when needed, and this is

better than nothing. If one is considering implementing full-scale ETA solution using opensource components it would require quite amount of time and effort.

Organization planning to do this should evaluate if they have enough competence and resources in responsible administrative team. What one can save in license costs using opensource product might be lost in administrative and operational costs.

Single node Security Onion installation would probably be suitable for many small organizations. It can be setup with minimal cost and effort and later on start to build more advanced detection capabilities by fine tuning it. Larger organizations might consider implementing Security Onion using distributed architecture, allowing it to scale for larger environments and possibly buy support contracts and dedicated hardware from Security Onion Solutions.

SensorFleet might be good solution for service provider or large organization which is considering implementing many sensors in many separate locations. SensorFleet however need SIEM or similar centralized component for doing analysis based on logs generated by SensorFleet.

When comparing opensource product to commercial ones, there are certain benefits in both. Opensource is obviously cheaper when comparing strictly costs for implementing system. However, administrative and operational costs in opensource products may sometimes become higher than licensing costs in commercial products. Commercial solution utilizing ML-based detection would require significantly less operational time from security analyst than opensource solution relying on rule-based detection.

One aspect to evaluate is the business criticality of the system. If organization is a service provider offering ETA services for their customers, the business criticality of the system is much higher than it would be for organization that performs basic ETA just for their own purpose. If organization is tied to high service level agreements, then reliability of the system and support contracts to vendors become more important. On the other hand, using opensource product allows system administrations to have visibility for everything what is going on in the system and makes it possible to customize and integrate the system with external systems without limitations that commercial systems might have. In the end, it comes to the

type of organization is and what is their business. If organization has SecDevOps functionality and ETA is business critical for them, opensource ETA might be better than commercial solutions. For small organizations where ETA is not business critical, simple opensource ETA solution might be suitable as well. For service providers and large organizations with decent budgets for ETA, commercial solution might be better fit than opensource products.

9 Discussion

Even though delineating subject of the thesis to ETA for TLS, it was still huge topic go through and some aspects of it couldn't be investigated as thoroughly as other. Overall, all aspects related to subjects were investigated, researcher learned a lot from the topic during thesis and research objectives were reached. During writing of the thesis, it came clear that many others in the field of science have been studying the same subject before, but since technology is constantly evolving and systems are being replaced by others, researcher thinks this research might give some additional value to science community.

During testing phase, it was clear that rule based detection has its place for detecting known IoCs. But role of that is diminishing in ETA as TLS 1.3 along with eSNI, ECH and DoH gains more popularity. When SNI and certificates cannot be seen in clear text anymore, signature based detection relying on them become useless. JA3/JA3s hashes can be used in the future as well, but with little customization for their tools, threat actors can quite easily avoid JA3/JA3s based detections. Behavior-based detection will become more important in the future. At the moment, there doesn't seem to be opensource solutions utilizing machine learning algorithms for behavior-based detection, which could challenge the commercial ones. This would be good subject for further researchs.

When it comes to automation, there is definitely a need for that as number of different NDR solutions out there indicates. But researcher still believes there is a place for threat hunting with human factor in it. After all, threat actors are humans and it takes another human to analyze how another human thinks what motives him.

Machines can detect IoCs and behavioural anomalies but tracking TTPs used by APTs cannot be automated.

References

Althouse, J 2019. TLS Fingerprinting with JA3 and JA3S. Article by John Althouse, Jan 15, 2019. Accessed on 21 May 2021. Retrieved from

<https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967>

ACSC. 2020. Australian Cyber Security Centre: Defending Against the Malicious Use of the Tor Network. Accessed on 15 March 2021. Retrieved from

<https://www.cyber.gov.au/acsc/view-all-content/publications/defending-against-malicious-use-tor-network>

Bianco, D. 2013. "Pyramid of Pain" blog post. Posted originally 1st March 2013 by David J. Bianco. Accessed 23 January 2021. Retrieved from [http://detect-](http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html)

[respond.blogspot.com/2013/03/the-pyramid-of-pain.html](http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html)

Chrisander, M 2020. A simple guide to Threat Hunting. Blog By Martha Chrisander, posted October 1st, 2020. Accessed on 18 May 2021. Retrieved from

<https://www.logpoint.com/en/blog/threat-hunting>

Dahan, A., Rochberger, L., Salem, E., Zhao, M., Yona, N., Yampel, O., Hart, M. 2019. Dropping Anchor: From a TrickBot Infection to the Discovery of the Anchor Malware". Accessed 14 May 2021. Retrieved from

<https://www.cybereason.com/blog/dropping-anchor-from-a-trickbot-infection-to-the-discovery-of-the-anchor-malware>

Darktrace, 2021. Darktrace AI: Combining Unsupervised and Supervised Machine Learning. Darktrace AI whitepaper. Accessed on 18 May 2021. Retrieved from

<https://www.darktrace.com/resources/wp-machine-learning.pdf>

Delaney, D. 2017. How to Passively Detect VPN Clients on Your Network. NetFort Blog By: Darragh Delaney 5 December 2017. Accessed on 15 March 2021. Retrieved from <https://www.netfort.com/blog/detect-vpn-clients-network>

Desai, D. 2017. Sophisticated Malware Strains Using SSL to Encrypt Activity. Blog post by Deepen Desai, 2 August 2017. Accessed on 11 March 2021. Retrieved from

<https://www.zscaler.com/blogs/security-research/ssltls-based-malware-attacks>

EC-Council, 2021. 2 Popular Cyber Threat Intelligence Feeds and Sources. EC-Council Blog post from 6 Sep 2020. Access on 18 May 2021. Retrieved from <https://blog.eccouncil.org/2-popular-cyber-threat-intelligence-feeds-and-sources>

ENISA. 2019. Encrypted Traffic Analysis: Use Cases and Security Challenges. Accessed on 16 October 2020. Retrieved from <https://www.enisa.europa.eu/publications/encrypted-traffic-analysis>

F5. 2020. Decrypt SSL and TLS 1.3 for Inspection and Protection. Accessed on 26 October 2020. Retrieved from <https://www.f5.com/resources/library/encrypted-threats/ssl-visibility/is-tls-1-3-the-solution#spa-2>

Farrel, C. 2018. Looking Under the Rock: Deployment Strategies for TLS Decryption. Accessed on 26 October 2020. Retrieved from <https://www.sans.org/reading-room/whitepapers/dlp/rock-deployment-strategies-tls-decryption-38240>

Flowmon, 2020. Encrypted Traffic Analysis: The data privacy-preserving way to regain visibility into encrypted communication. Accessed on 27 November 2020. Retrieved from <https://www.flowmon.com/en/resources/encrypted-traffic-analysis>

Gartner, 2020. Market Guide for Network Detection and Response. Published 11 June 2020. Retrieved from <https://www.gartner.com/en/documents/3986225>

Goddard, N. 2020. Getting Started on Contributing to RITA. Article by Naomi Goddard published November 11, 2020. Accessed on 17 May 2021. Retrieved from <https://www.activecountermeasures.com/getting-started-on-contributing-to-rita>

Hjelm, D. 2019. A New Needle and Haystack: Detecting DNS over HTTPS Usage. Research paper by Drew Hjelm September 10, 2019. Accessed on 15 March 2021. Retrieved from <https://www.sans.org/reading-room/whitepapers/dns/paper/39160>

Henderson, J., Hubbard, J. 2018. SANS Webcast: Prioritizing Log Enrichment. Recorded November 06, 2018. Accessed on 17 April 2021. Retrieved from <https://www.sans.org/webcasts/prioritizing-log-enrichment-109275>

IANA, 2019. IANA IPv4 Special-Purpose Address Registry. Accessed on 13 May 2021. Retrieved from <https://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml>

Kaspersky, 2021a. How to get rid of malware? Article on Kaspersky's web site. Accessed on 11 March 2021. Retrieved from <https://www.kaspersky.com/resource-center/threats/malware-protection>

Kaspersky, 2021b. What is a Botnet? Article on Kaspersky's web site. Accessed on 11 March 2021. Retrieved from <https://www.kaspersky.com/resource-center/threats/botnet-attacks>

Kaspersky, 2021c. Brute Force Attack: Definition and Examples. Article on Kaspersky's web site. Accessed on 12 March 2021. Retrieved from <https://www.kaspersky.com/resource-center/definitions/brute-force-attack>

LogPoint, 2021. User and Entity Behavior Analytics (UEBA): Accelerated detection and response. Article on LogPoint web site. Accessed on 18 May 2021. Retrieved from <https://www.logpoint.com/en/product/ueba-solution>

Messier, R. 2017. Network Forensics. 10475 Crosspoint Boulevard, Indianapolis, IN 46256: John Wiley & Sons, Inc.

Nagy, L. 2020. Sophos News: Nearly a quarter of malware now communicates using TLS. Published 18 February 2020. Accessed on 11 March 2021. Retrieved from <https://news.sophos.com/en-us/2020/02/18/nearly-a-quarter-of-malware-now-communicates-using-tls/>

NIST, 2019. NIST Special Publication 800-52 Revision 2: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. Accessed on 18 December 2020. Retrieved from <https://doi.org/10.6028/NIST.SP.800-52r2>

Nohe, P. 2019a. TLS 1.3: Everything you need to know. Blog post on www.thesstore.com July 16, 2019. Accessed 4 December 2020. Retrieved from <https://www.thesstore.com/blog/tls-1-3-everything-possibly-needed-know>

Nohe, P. 2019b. Cipher Suites: Ciphers, Algorithms and Negotiating Security Settings. Blog post on www.thesstore.com. Accessed on 5 January 2021. Retrieved from

<https://www.thesslstore.com/blog/cipher-suites-algorithms-security-settings>

Kusek, C. 2019. "Darktrace - What is it?" video on Youtube presented by Xiologix CTO Christopher Kusek. Accessed on 27 November 2020. Retrieved from

<https://www.youtube.com/watch?v=5387TV6rLdY>

Marstrander, M., Malvica, M. 2020. SNIcat: Circumventing the guardians. Mnemonic Labs blog post 8 December 2020. Accessed 12 March 2021. Retrieved from

<https://www.mnemonic.no/blog/introducing-snicat>

McClurg, J 2020. Decade of the RATs: Novel Cross-Platform APT Attacks Targeting Linux Windows and Android. Article by John McClurg, May 1 2020. Accessed 12 March 2021. Retrieved from <https://www.securitymagazine.com/articles/92253-decade-of-the-rats-novel-cross-platform-apt-attacks-targeting-linux-windows-and-android>

<https://www.securitymagazine.com/articles/92253-decade-of-the-rats-novel-cross-platform-apt-attacks-targeting-linux-windows-and-android>

Oakley, C. 2020. Nettitude Blog: The SOC Visibility Triad – SIEM, EDR & NDR.

Accessed 26 October 2020. Retrieved from

<https://blog.nettitude.com/the-soc-visibility-triad>

Pagano, E. 2020. "Protect Yourself From Phishing" blog posted March 25, 2020 by Elizabeth Pagano. Accessed 11 March 2021. Retrieved from

<https://www.ssl.com/guide/protect-yourself-from-phishing>

Patton, C. 2020. "Good-bye ESNI, hello ECH!" Cloudflare Blog posted 08/12/2020 by Christopher Patton. Accessed on 23 January 2021. Retrieved from

<https://blog.cloudflare.com/encrypted-client-hello>

RFC 5288. 2008. AES Galois Counter Mode (GCM) Cipher Suites for TLS. Accessed on 11 December 2020. Retrieved from <https://tools.ietf.org/html/rfc5288>

RFC 7539. 2015. ChaCha20 and Poly1305 for IETF Protocols. Accessed on 11 December 2020. Retrieved from <https://tools.ietf.org/html/rfc7539>

RFC 8446. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. Accessed on 13 November 2020. Retrieved from <https://tools.ietf.org/html/rfc8446#section-1.2>

<https://tools.ietf.org/html/rfc8446#section-1.2>

RFC 8484. 2018. DNS Queries over HTTPS (DoH). Accessed on 15 March 2021.

Retrieved from <https://tools.ietf.org/html/rfc8484>

RITA, 2021. RITA: Real Intelligence Threat Analytics. Accessed on 17 May 2021.

Retrieved from <https://www.activecountermeasures.com/free-tools/rita>

Rezek, M. 2020. What is the difference between signature-based and behavior-based intrusion detection systems? Article by Michael Rezek from December 9, 2020.

Accessed on 18 May 2021. Retrieved from <https://accedian.com/blog/what-is-the-difference-between-signature-based-and-behavior-based-ids>

Ristić, I. 2018. Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications. London: Feisty Duck

Santos, O. 2020. CCNP and CCIE Security Core SCOR 350-701 Official Cert Guide.

Hoboken, New Jersey: Cisco Press.

Security Onion, 2021. Security Onion 2.3 Documentation. Accessed on 16 April 2021.

Retrieved from <https://docs.securityonion.net/en/2.3>

Security Onion Solutions, 2021. Security Onion Solutions web page. Accessed on 17

May 2021. Retrieved from <https://securityonionsolutions.com>

SensorFleet, 2021a. SensorFleet web page. Accessed on 17 May 2021. Retrieved

from <https://sensorfleet.com>

SensorFleet, 2021b. SensorFleet version 2.3.0 User Manual.

Shaw, C. 2020. How to Identify Malicious Network and Port Scanning. Blog post by

Christine Shaw on December 19, 2020. Accessed on 15 March 2021. Retrieved from

<https://www.extrahop.com/company/blog/2016/how-to-recognize-malicious-network-scanning-port-scanning>

Skerrit, B. 2020. How Does Tor Really Work? The Definitive Visual Guide. Blog post by

Brandon Skerritt September 19, 2020. Accessed on 15 March 2021. Retrieved from

<https://skerritt.blog/how-does-tor-really-work>

SSL Labs, 2021. SSL Labs website: SSL Pulse Monthly Scan. Accessed on 21 May 2021.

Retrieved from <https://www.ssllabs.com/ssl-pulse/>

- Strand, J. 2020. Detecting Malware Beacons With Zeek and RITA. Black Hills Information Security blog by John Strand, 3 Mar 2020. Accessed on 17 May 2021. Retrieved from <https://www.blackhillsinfosec.com/detecting-malware-beacons-with-zeek-and-rita>
- Sullivan, N. 2018. A Detailed Look at RFC 8446 (a.k.a. TLS 1.3). Blog post on coudflare. Accessed on 5 January 2021. Retrieved from <https://blog.cloudflare.com/rfc-8446-aka-tls-1-3>
- Suricata, 2021a. Suricata documentation: latest version. Accessed on 15 March 2021. Retrieved from <https://suricata.readthedocs.io/en/latest/>
- Suricata, 2021b. Suricata: Open Source IDS / IPS / NSM engine. Accessed on 6 May 2021. Retrieved from <https://suricata-ids.org/>
- Tolbert, J 2020. Leadership Compass: Network Detection and Response. KuppingerCole report by John Tolbert June 10, 2020. Accessed 2 April 2021. Retrieved from <https://plus.kuppingercole.com/article/lc80126/network-detection-and-response>
- Yu, D 2020. Network Detection and Response: The Building Blocks. Blog post by David Yu July 7, 2020. Accessed on 16 April 2021. Retrieved from <https://www.hillstonenet.com/blog/network-detection-and-response-the-building-blocks>
- Zeek, 2021a. Zeek Documentation for LTS version (v4.0.1). Accessed on 16 April 2021. Retrieved from <https://docs.zeek.org/en/lts>
- Zeek, 2021b. Zeek: An Open Source Network Security Monitoring Tool. Accessed on 6 May 2021. Retrieved from <https://zeek.org>
- Zimba, A., Mulenga, M. 2018. A Dive into the Deep: Demystifying WannaCry Crypto Ransomware Network Attacks Via Digital Forensics. Accessed on 3 March 2021. Retrieved from https://www.researchgate.net/publication/325678210_A_Dive_into_the_Deep_Demystifying_WannaCry_Crypto_Ransomware_Network_Attacks_Via_Digital_Forensics

Appendices

Appendix 1. NIST recommended ciphersuites

TLS Version	Certificate Type	Ciphersuite
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2B)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2C)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_128_CCM (0xC0, 0xAC)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_CCM (0xC0, 0xAD)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (0xC0, 0xAE)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 (0xC0, 0xAF)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x23)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x24)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA21 (0xC0, 0x09)
1.2	ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xC0, 0x0A)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2F)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)
1.2	RSA	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x00, 0x9E)
1.2	RSA	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x00, 0x9F)
1.2	RSA	TLS_DHE_RSA_WITH_AES_128_CCM (0xC0, 0x9E)
1.2	RSA	TLS_DHE_RSA_WITH_AES_256_CCM (0xC0, 0x9F)
1.2	RSA	TLS_DHE_RSA_WITH_AES_128_CCM_8 (0xC0, 0xA2)
1.2	RSA	TLS_DHE_RSA_WITH_AES_256_CCM_8 (0xC0, 0xA3)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x27)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x28)
1.2	RSA	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x00, 0x67)
1.2	RSA	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x00, 0x6B)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x13)
1.2	RSA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x14)
1.2	RSA	TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x00, 0x33)
1.2	RSA	TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x00, 0x39)
1.2	DSA	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 (0x00, 0xA2)
1.2	DSA	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 (0x00, 0xA3)
1.2	DSA	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 (0x00, 0x40)
1.2	DSA	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 (0x00, 0x6A)
1.2	DSA	TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x00, 0x32)
1.2	DSA	TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x00, 0x38)
1.2	DH	TLS_DH_DSS_WITH_AES_128_GCM_SHA256 (0x00, 0xA4)
1.2	DH	TLS_DH_DSS_WITH_AES_256_GCM_SHA384 (0x00, 0xA5)
1.2	DH	TLS_DH_DSS_WITH_AES_128_CBC_SHA256 (0x00, 0x3E)
1.2	DH	TLS_DH_DSS_WITH_AES_256_CBC_SHA256 (0x00, 0x68)
1.2	DH	TLS_DH_DSS_WITH_AES_128_CBC_SHA (0x00, 0x30)
1.2	DH	TLS_DH_DSS_WITH_AES_256_CBC_SHA (0x00, 0x36)
1.2	DH	TLS_DH_RSA_WITH_AES_128_GCM_SHA256 (0x00, 0xA0)
1.2	DH	TLS_DH_RSA_WITH_AES_256_GCM_SHA384 (0x00, 0xA1)
1.2	DH	TLS_DH_RSA_WITH_AES_128_CBC_SHA256 (0x00, 0x3F)
1.2	DH	TLS_DH_RSA_WITH_AES_256_CBC_SHA256 (0x00, 0x69)
1.2	DH	TLS_DH_RSA_WITH_AES_128_CBC_SHA (0x00, 0x31)
1.2	DH	TLS_DH_RSA_WITH_AES_256_CBC_SHA (0x00, 0x37)

1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x2D)
1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x2E)
1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x25)
1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x26)
1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xC0, 0x04)
1.2	ECDH	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xC0, 0x05)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 (0xC0, 0x31)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x32)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 (0xC0, 0x29)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 (0xC0, 0x2A)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xC0, 0x0E)
1.2	ECDH	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xC0, 0x0F)
1.3	RSA or ECDSA	TLS_AES_128_GCM_SHA256 (0x13, 0x01)
1.3	RSA or ECDSA	TLS_AES_256_GCM_SHA384 (0x13, 0x02)
1.3	RSA or ECDSA	TLS_AES_128_CCM_SHA256 (0x13, 0x04)
1.3	RSA or ECDSA	TLS_AES_128_CCM_8_SHA256 (0x13, 0x05)

Appendix 2.

TLS keywords used in Suricata version 6

Suricata Keyword	Description
tls.cert_subject	Match TLS/SSL certificate Subject field.
tls.cert_issuer	Match TLS/SSL certificate Issuer field.
tls.cert_serial	Match on the serial number in a certificate.
tls.cert_fingerprint	Match on the SHA-1 fingerprint of the certificate.
tls.sni	Match TLS/SSL Server Name Indication field.
tls_cert_notbefore	Match on the NotBefore field in a certificate.
tls_cert_notafter	Match on the NotAfter field in a certificate.
tls_cert_expired	Match returns true if certificate is expired.
tls_cert_valid	Match returns true if certificate is not expired.
tls.certs	Do a "raw" match on each of the certificates in the TLS certificate chain
tls.version	Match on negotiated TLS/SSL version.
ssl_version	Match version of SSL/TLS record.
tls.subject	Match TLS/SSL certificate Subject field.
tls.issuerdn	match TLS/SSL certificate IssuerDN field
tls.fingerprint	match TLS/SSL certificate SHA1 fingerprint
tls.store	store TLS/SSL certificate on disk
ssl_state	The ssl_state keyword matches the state of the SSL connection. The possible states are client_hello, server_hello, client_keyx, server_keyx and unknown

Appendix 3. Virtual machines used in testing environment

Hostname	IP address	Operating System	Softwares	Role
kali	203.0.113.31	Kali 2021.1	Kali tools	Threat Actor
www.fzecure.com	203.0.113.13	Centos 7	NGINX 1.16.1	Phishing Site
firewall	10.111.1.1 10.111.100.1 10.222.0.89 203.0.113.1	OpenWRT 15.05		Firewall and router
db.int.lupari.fi	10.111.1.44	Centos 7	NGINX 1.16.1	Internal Server
intra.int.lupari.fi	10.111.1.55	Centos 7	NGINX 1.16.1	Internal Server
windows10	10.111.100.100	Windows 10		Workstation
ubuntu18	10.111.100.111	Ubuntu 18.04		Workstation
SensorFleet Manager	10.222.0.80 10.222.0.88	Ubuntu 18.04	IDS Policy Manager, Downloader	ETA1 SensorFleet Manager
SensorFleet Sensor	10.222.0.81 10.222.0.88	Ubuntu 18.04	Capture Engine, Zeek, Suricata, Logforwarder	ETA1 SensorFleet Sensor
Security Onion	10.222.0.83		Security Onion	ETA2 Collector and analyzer
RITA	10.222.0.86	Ubuntu 18.04	RITA	ETA2 C2 analyzer
LogPoint	10.222.0.186		LogPoint	ETA1 SIEM/analyzer