**Ray’s Original Network Intrusion Detection Systems: Now in Nacho Cheese and Cool Ranch**

The technological innovations surrounding the field of cybersecurity are varied and numerous. A primary component of a strong security stance is monitoring activity on your network in the event your perimeter defenses have been undermined. One of the most commonly used tools is the Network Intrusion Detection System, hereinafter referred to as NIDS. This tool is used to analyze traffic to ensure that no packets are being sent into a network that may compromise operations whether they be used for malicious code execution or through violation of network policies. NIDS comprise a very specific niche in network security. They do not prevent threats from accessing a network, nor do they have any ability to respond to the threat, they only report alerts to administrators when they detect suspicious traffic. NIDS can be categorized based on their basic operating structure.

It is important to understand the ways in which NIDS function before we can discuss the ways used to circumvent them. While NIDS is a broad category of security tools, there are a few divisions within that group based on how the specific tool functions. NIDS can be placed into subgroups based on whether they are real time vs stored data analysis and whether they are signature-based vs anomaly-based analysis. NIDS that analyze traffic in real time can scan packets as they travel between hosts and alert an administrator as soon as it detects specific traffic. On the other hand, a NIDS that analyze traffic after it has been logged and stored are able to run the data through a variety of processes which allows for a deeper analysis in the hopes of presenting a more accurate event report to administrators. [Should we put something about where and how an IDS is placed in a network?]

A third NIDS method is called stateful protocol analysis and includes similar behavior of both anomaly and signature-based detections systems. Stateful protocol analysis works by profiling and establishing baseline behavior, and flags deviations to this protocol. Protocol analysis signatures can also be designed to overcome attempts by attackers to obfuscate their exploits.....

The second division of NIDS can be found in how they analyze network traffic. Signature based NIDS analyze incoming network traffic by comparing it to databases that keep records of known attacks. These systems rely on an up to date library in order to be effective, as the system cannot make accurate assessments on traffic that does not meet the specific criteria found in that library.

The second variety is anomaly based, which can analyze the baseline traffic and can pick the outliers from the traffic to alert administrators to suspicious occurrences.[1] These systems work using a machine learning algorithm which can be heuristically refined by the administrator making final judgement calls on alerts they receive. The ultimate goal of these algorithms is to maximize correct classifications and minimize incorrect classifications but, much like humans, they will always make some mistakes in judgement. Both of these operating structures have critical blind spots that attackers can exploit in order to evade detection when sending data through a network.

Signature based detection analyzes incoming traffic and compares it to known signatures in its database. If a signature matches a file in the database, then this match is marked as a threat and an alert is raised. This type of system may include alerts of anything that it deems malicious, such as Command & Control (C2), malware payloads, or exfiltration of valuable data.The biggest downsides to Signature based detection is that if the signature does not exist in the database then it can become compromised because it cannot account for attack behavior that is not already preprogramed in its list of what it deems as a threat.

A prime example of a vendor tool for signature type detection is Snort. Per Snort’s own description, they are an open source intrusion prevention system capable of real-time traffic analysis and packet logging. Snort comes in three different types of modes: IDS Mode, Logging Mode, and Sniffer Mode. So, how does Snort classify threats? Well, it contains a file named snort.conf which lives inside of the `etc/` directory of Snort and in this conf file are all the preprocessor and rule files which it uses to compare packets to and if it matches a rule inside this file then it’s identified as a threat. Below is a snippet of what the snort.conf rules look like. (Figure 1)

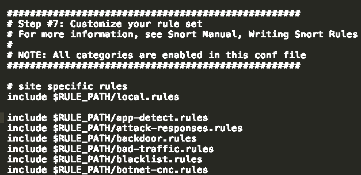


Figure 1 [Source]

Snort is considered a very powerful tool, a tool in which you can write about every rule imaginable. While that seems like the greatest thing it comes at a price, that price is having a dedicated team with the knowledge to write out every rule that would apply to your network setup.

The detail that goes into writing each rule might be nuanced enough to be the downfall of this tool. “With great power comes great responsibility”. The responsibility of capturing thousands upon thousands of rules that you might never be able to catch up with. Especially with new attacks always on the rise and the need for new signatures to evade them.

But aside from there not being a signature for an attack, or creating enough rules to cover everything, another evasion technique that can be used against SNORT, or any NIDS tool for that matter, is the ability to exploit the flowbits feature.

**Flowbits Evasion**

So, what is a Flowbit? Well it is a flag that can be set by a rule and then used by another one. In other words, “It allows the detection engine to track state across a single TCP session. The support of stateful signatures allows a signature-based IDS to detect multi-stage attacks (An Evasive Attack on Flowbits).” A flowbit rule can be evaded if it can be triggered by the attacker to change an in-Snort session state while preserving the actual session state. [[An Evasive Attack on SNORT Flowbits]](https://www3.cs.stonybrook.edu/~ttran/noms2012.pdf)

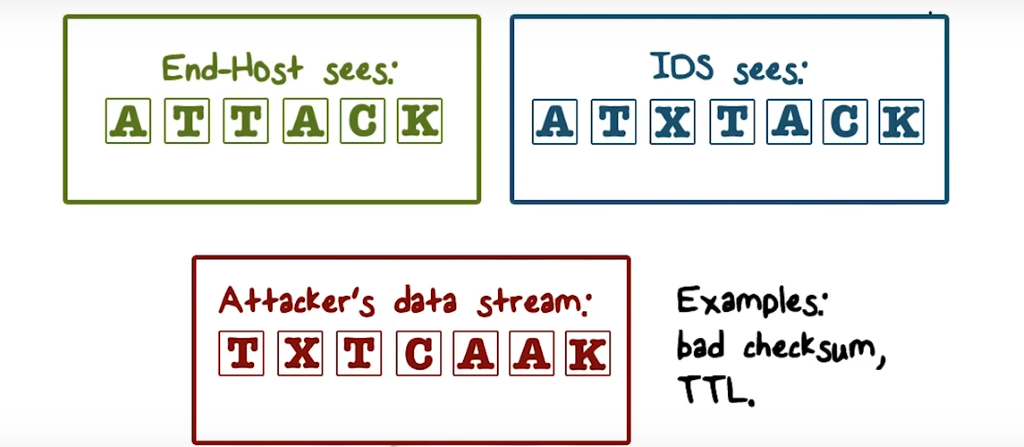
[NEED MORE IN DEPTH ON THIS]

Below are the most common types of attacks against NIDS.

**Obfuscation**

**Insertion Attack**

This type of attack tricks the IDS. It inserts an invalid packet like in the example below with the packet “X” and while the End Host can detect it the IDS doesn’t thus giving different streams to the IDS and target hosts ([Evading NIDS, revisited](https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=0e0383b5-1e34-42e0-afaa-65b0cadffd6e&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments)). This type of attack occurs when NIDS is less strict in processing packets than the internal network.



**Figure 3**

Another similar attack to the Insert attack method is Evasion, in which an attacker will send different packets just like the figure above but in this case instead of the End Host rejecting the packet, it is the IDS that rejects the packet and the End Host which accepts it, giving different streams yet again to the IDS and End Host. Evasion occurs due to NIDS being stricter in processing packets than the internal network thus allowing the End Host to accept packets that were rejected by NIDS.

But for any of these two techniques to be successful the attacker must also use something called packet fragmentation which is when an attack stream is broken down into smaller ones. These two types of techniques mostly exploit network and protocol ambiguities at the NIDS. The vagueness in the way in which it interprets the header field, handles the header options, and reassembles the fragments. Some examples of these ambiguities are TTL, Data, Length, and IP Frag Offset.

**Fragmentation**

There are two type of fragmentation methods, one which will overwrite a section of the previous fragment and the other method which will completely overwrite a fragment.

In both of these methods, an attacker sends malicious traffic that is split into such small pieces that it does not trigger the NIDS; this is successful against NIDS that don’t reconstruct packets before checking them against the signature rules.

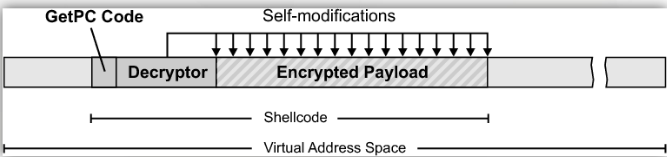
Another way to use this method is by sending small packet pieces over a long period of time in order to avoid an IDS “event horizon” or monitoring period in which an IDS will look for subsequent signatures that match an attack. Yet nother way to use fragmentation is sending the packets out of the order to confuse the simple packet re-assemblers but not the End Host computer ([Evasion\_Techniques\_Sneaking](http://speed.cis.nctu.edu.tw/~ydlin/pdf/Evasion_Techniques_Sneaking_through_Your_Intrusion_Detection_Prevention_Systems.pdf)).

**EXAMPLES OF ABOVE**

**Polymorphic Shellcode**

Another obfuscation method is Polymorphic Shellcode, which allows the attacker to maintain the original algorithm in place while simultaneously mutating to create a unique pattern that won’t be found in the IDS system. This method works well to avoid SNORT rules as well as other NIDS tools.

The way in which this technique works is by encrypting strings or malicious code inside of the shellcode. Due to the NIDS signature rules looking only for commonly used strings, the attacker can evade without being detected. Not only is the bad code encrypted but the common strings are as well.



Due to there being many ways to mutate a shellcode, detecting polymorphic shellcode becomes hard. “An IPS may need to decrypt the encrypted code to restore the original signature, or even emulate the code execution (e.g., on a sandbox that emulates the execution on the target hosts) to find malicious behavior. The tasks of restore the shell code semantics on-line are therefore computationally expensive and burden the load of an IPS. ([Evasion\_Techniques\_Sneaking\_through](http://speed.cis.nctu.edu.tw/~ydlin/pdf/Evasion_Techniques_Sneaking_through_Your_Intrusion_Detection_Prevention_Systems.pdf)).

**Pattern Matching Evasion**

**Application Hijacking**

**Solutions**

lksadfljk;asdfkl;jsdaflk;jsdf


Graphic illustrating basic function of an ML algorithm. An anomaly-based IDS will use its learning algorithm to make decisions on whether data packets meet expected criteria and, if not, will alert an administrator. As you can see in this model, there is a margin of acceptable loss, or incorrect judgments. [2]

A principal flaw in anomaly-based detection systems is that one can employ the same technology that makes the system effective to completely invalidate it. If the system uses trends in normal network traffic to find outliers, then why not use that same model to ensure your malicious traffic is able to match the traffic that would be found on the network? In order to do this, some researchers and malicious actors have been using machine learning algorithms to create their own “counter-IDS,” if you will. The concept of using machine learning against a machine learning algorithm is known as “adversarial machine learning.”[1] In a study presented at the 2020 Annual Conference on Information Sciences and Systems, researchers were successfully able to evade a machine learning NIDS algorithm by running an adversarial model to create “target misclassification” and cause the detection system to classify malicious activity as benign.

In the study, the team trained a machine learning algorithm using commonly available datasets including real world network attacks until it achieved an accuracy of over 99% in detecting attacks. They then used a Jacobian Saliency Map Attack(JSMA) to create “perturbation” in the same dataset samples and compared the detection results.[1] In layman’s terms, a JSMA uses a mathematical formula to predict what parts of a data sample are weighted most importantly in making a classification, in this case “attack” or “benign”, and spikes one in order to throw the weighted average and create a change in classification.[4] In the experiment, using this technique showed a decrease in accuracy of roughly 25% for predicting whether a data sample was an attack. [1]

Despite this research being done in a “white-box” setting, meaning the malicious party had complete knowledge of the inner workings of its target, it is not inconceivable to apply these strategies in a “black-box” setting. It would be simple enough for a well-funded actor to acquire licenses for the NIDS product they are attempting to bypass and either mine the code or run widely used datasets through the product to make a determination about what type of algorithms are applied to it. In fact, it has been demonstrated that information regarding a machine learning algorithms model can be extracted using prediction APIs by looking at confidence values in algorithm trees.[3] Once they have "stolen” the model, they would be able to create attacks against it to see if they are able to circumvent the system.

Of course, the natural response to this threat would be for defenders to recognize the requirement for taking adversarial learning into account. To do this, researchers have tested the effectiveness of teaching detection software to detect adversarial learning. They found that while they could reliably detect adversarial attacks, they had a massive increase in false positives. [1] This is understandable as you are basically asking your machine to always assume that the attack is just outside of its classification parameters. Since machine learning was not originally conceived of to operate in an adversarial manner, it is possible that, with further research, adversarial learning will be better accounted for.

Another fundamental flaw in the anomaly-based detection system is its tendency to have a high number of false positives. In a perfect world, security team members would be able to respond to every alert with equal care and attention but, in reality, there are simply too many tasks in a day to work on and you have to decide where your resources are best spent. Since their inception, anomaly based detection systems have been derided due to the sometimes thousands of alerts they can put out in a day, many of them being false alarms. [10] Each of these alerts then must be investigated individually by analysts to determine their validity. This can quickly lead to alert fatigue, in which analysts begin to ignore the intrusion detection system, thereby undermining its usefulness and allowing real attacks to get through. The central conceit that anomaly based detection was built on, was that networks have a “baseline” traffic behavior that can then be used to compare against new network traffic to find potential attacks. While this may be producible in a testing environment, it is not a realistic expectation for real world deployment. Creating a reliable snapshot of “normal” traffic relies on not only a massive dataset, but a massive dataset over a very long time. Even once this is gathered, it still cannot account for the changes found in a network, especially one at enterprise scale where this type of technology is most likely used. Network demands fluctuate, employees come and go, and new devices are added and reconfigured every day. This is not even accounting for the widespread adoption of WFH and BYOD that we are seeing currently. Simply put, the only way to have a reliably high intrusion detection rate is to also accept a very high false positive rate. [9] Some modern companies, for example Darktrace, claim that they have found a way around this by adding “deep learning” into their algorithms. Administrators that actually work with the software, however, have found it to be plagued with the same problems as anomaly based systems that came before it. [11]

A final example of an attack on the very foundation of anomaly based detection is very similar to the concept of adversarial learning but has to do with user behavior. In a world of increasingly complex authentication policies, attackers are increasingly looking to steal legitimate credentials to conduct their operations. It also highlights the fact that some malicious network traffic can look just like legitimate traffic, making it very difficult to identify.

It is expected that an attacker with compromised credentials will likely not know the network architecture and will act erratically as they attempt to move laterally towards a target. Some anomaly-based intrusion detection systems are designed to alert on this unusual behavior. In a study conducted by researchers from the University of Lima and Indra Digital Labs, it was demonstrated how an attacker could bypass detection by mimicking the behavior of a normal user while still attempting to engage in malicious behavior. They demonstrated this through three separate strategies: obfuscation, action pruning, and noise generation. Obfuscation simply entails performing legitimate user actions in between malicious actions in order to water down what would normally qualify as alert criteria. Action pruning is slightly more difficult as it requires the attacker to have some understanding of the monitoring time periods used by the detection system. Detection systems can employ a timed period, called an “event horizon”, in which they look for subsequent network actions to determine if an attack is taking place. By reducing the number of actions an attacker takes during each monitoring period, they can continue to operate without raising alerts. Noise generation is effectively a combination of the previous two categories where an attacker pads malicious activity with normal activity in a monitoring period. Using these techniques, they were able to reduce detection rates significantly, with an corresponding decrease in detection for number of actions observed. This demonstrates the lack of utility of anomaly-based systems when facing a skillful, long-term attack.

A possible defense against such an attack would be tuning monitoring period intervals in such a way that it makes attacks relying on this behavior impractical and cumbersome. Shortening the event horizon to get a more granular view of malicious behavior sequences would make it harder to obfuscate malicious actions. Lengthening the event horizon, would be much more computationally expensive, but it could be done while tightening up the alerting parameters to make action pruning unfeasible. Either way, it must accept that both of these strategies would lead to an increase in false positive alerts caused by legitimate users.

One particularly effective method of disguising malicious traffic involves the utilization of encryption.

**Encrypted Evasion**

[transition] if network traffic is encrypted, NIDS cannot analyze it. Attackers use this to their advantage by encrypting their own malicious traffic and thereby completely evading deep packet inspection (7 Roques). TLS-based threats allow bad actors to deposit payloads, exfiltrate data, obfuscate C2 commands, and of course, execute phishing schemes more believably. A notable example of this includes the financial trojan Dridex, a banking malware that relies on TLS encryption to both download and successfully communicate with its C2 server. What is interesting to note is that Dridex uses a recent version of TLS 1.2 with several extensions and cipher suites. The packet in Figure B1 shows that the server sent a self-signed certificate, which should automatically raise red flags, however; it is very easy for threat actors to use a legitimate service such as Let’s Encrypt to obtain valid certificates.

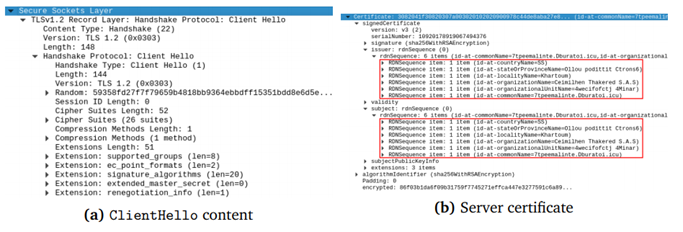


Figure B1: Dridex “Client Hello” packet. Note, the only abnormality would be a self-signed certificate.

An attempt to combat malicious encrypted connections led to the development of JA3 TLS. It is important to note that while TLS traffic at the application layer is encrypted, TLS handshakes are transmitted in the clear. Therefore, it is possible to view “Client Hello” packets such as that shown above. What JA3 does is take these packets and concatenates the Version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve Formats, into a unique md5-hashed fingerprint. As John Althouse explains, “In the event that a threat actor custom-built their own malware executable, it’s likely that the JA3 fingerprint will be unique to that executable” (“TLS Fingerprinting with JA3 and JA3S”). An example of one such detectable payload would be TrickBot which has the fingerprint of 1aa7bf8b97e540ca5edd75f7b8384bfa. Furthermore, there exists multiple databases of malicious JA3 fingerprints such as ja3er.com and sslbl.abuse.ch/ja3-fingerprints. One would assume that these can be used to blacklist known threats. Putting this into practice, these additional indicators of behavior can be essential for network and security operations teams to identify malware and protect their assets (“JA3 Fingerprinting: Encrypted Threat Detection”). As with any signature-based detection system, however, JA3 fingerprinting detection has the same inherent drawbacks.

By relying on a database of fingerprinted threats, JA3 “shares the same limitations of all other defenses that rely on pre-identified threats or blacklists” (Heinemeyer, “Darktrace”). This means that it is effectively null against unknown exploits, or zero-days. Additionally, though Salesforce touts that JA3 fingerprints remain constant throughout changes in IPs, ports, and X509 certificates, there are actually numerous reasons a JA3 signature can vary, as shown in Figure B2. Thus, not only are signatures malleable, they are also susceptible to tampering.

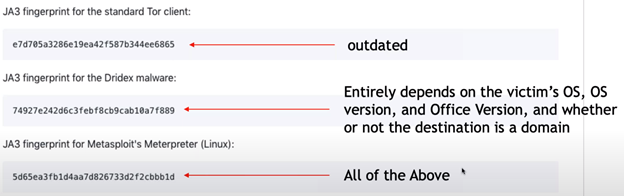


Figure B2: (JA)3 Reasons to Rethink Your Encrypted Traffic Analysis Strategies Webinar

In altering a specific component of a packet that JA3 uses to generate an MD5 fingerprint, one can effectively evade signature detection. One example would be changing the Cipher Suites, or as the Akami Threat Research calls it, Cipher Stunting. In this advanced example, bots were observed to be randomizing SSL/TLS signatures at a rate never seen before. As Zioni explains, “Those responsible are presenting a randomized cipher suite list in the 'Client Hello' messages, that in turn, randomize the hashes at the end. This is due to the relatively small and finite set of the SSL/TLS stack implementations available today. Each one allows for a different level of user intervention and customization of the SSL/TLS negotiation.” (“Bots Tampering with TLS to Avoid Detection”). Simply put, threat actors can evade detection by tinkering with TLS signatures. In conjunction with encryption-based techniques, phishing becomes a more powerful and convincing exploit in fooling the Layer 8 problem, i.e. users.

**Exploiting Trusted Services**

Using a combination of trusted services with any of the previously mentioned evasion techniques, threat actors can completely negate NIDS detection to exploit the weak and fallible Layer 8. A 2019 FireEye Email Threat Report indicated a dramatic increase of malicious payloads via trusted file hosting services, the most used being Dropbox (“FireEye 2019 Threat Report”). Though there are services that use CASBs, Cloud Access Security Brokers, this layer of security is not designed to detect threats such as malicious files, URLs, and social engineering attempts (Reich). The same FireEye research saw a 26% increase in malicious URLs using HTTPS, fooling the Layer 8 into believing they are on a secure connection. Furthermore, the domain of a malicious link can be obfuscated via domain fronting.

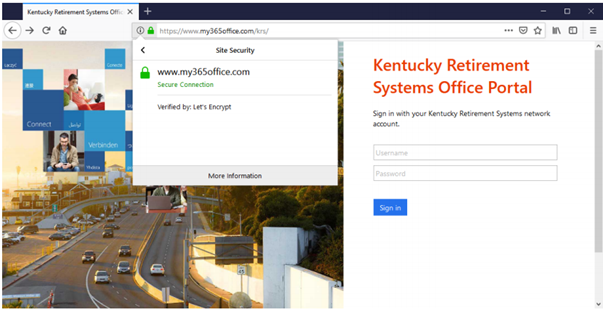


Figure B3: A convincing phishing website with a green padlock seal of approval

By amalgamating the techniques of TLS encryption, leveraging a trusted 3rd party for C2, and hiding that C2 traffic behind encryption and domain fronting, it should not come as a complete surprise that phishing attacks remain among the top threat action varieties (2020 DIBR).

It is expected that an attacker with compromised credentials will likely not know the network architecture and will act erratically as they attempt to move laterally towards a target. Some anomaly-based intrusion detection systems are designed to alert on this unusual behavior. In a study conducted by researchers from the University of Lima and Indra Digital Labs, it was demonstrated how an attacker could bypass detection by mimicking the behavior of a normal user while still attempting to engage in malicious behavior. They demonstrated this through three separate strategies: obfuscation, action pruning, and noise generation. Obfuscation simply entails performing legitimate user actions in between malicious actions in order to water down what would normally qualify as alert criteria. Action pruning is slightly more difficult as it requires the attacker to have some understanding of the monitoring time periods used by the detection system. Detection systems can employ a timed period, called an “event horizon”, in which they look for subsequent network actions to determine if an attack is taking place. By reducing the number of actions an attacker takes during each monitoring period, they can continue to operate without raising alerts. Noise generation is effectively a combination of the previous two categories where an attacker pads malicious activity with normal activity in a monitoring period. Using these techniques, they were able to reduce detection rates significantly, with an corresponding decrease in detection for number of actions observed. This demonstrates the lack of utility of anomaly-based systems when facing a skillful, long-term attack.

A possible defense against such an attack would be tuning monitoring period intervals in such a way that it makes attacks relying on this behavior impractical and cumbersome. Shortening the event horizon to get a more granular view of malicious behavior sequences would make it harder to obfuscate malicious actions. Lengthening the event horizon, would be much more computationally expensive, but it could be done while tightening up the alerting parameters to make action pruning unfeasible. Either way, it must accept that both of these strategies would lead to an increase in false positive alerts caused by legitimate users.

IN CONCLUSION.... there are many evasive techniques employable to defeat both signature and anomaly-based Network Intrusion Detection Systems. By themselves, NIDS and the various systems designed to improve on them, in preventing an attack. It must be combined with prevention systems, and is most effective when operated by a highly trained security professional. The most important takeaway from this paper is that you can never place too much trust in a security system. No system is an ironclad defense from a determined attacker, no matter what marketing literature may have you believe. The second most important takeaway is that attackers can be just as resourceful and innovative as defenders and so you must always keep alert and stay apprised of the new developments in the field.

References

1. https://www.researchgate.net/publication/340998746\_Model\_Evasion\_Attack\_on\_Intrusion\_Detection\_Systems\_using\_Adversarial\_Machine\_Learning
2. https://www.cray.com/blog/role-graph-data-appliance-machine-learning/
3. https://www.researchgate.net/publication/308027534\_Stealing\_Machine\_Learning\_MoMode\_via\_Prediction\_APIs
4. https://arxiv.org/pdf/1808.07945.pdf
5. https://www.securityweek.com/encrypted-network-traffic-comes-cost [Part of decryption PAN-OS too expensive]
6. <https://www.theseus.fi/bitstream/handle/10024/226340/Korhonen_Joni.pdf?sequence=2&isAllowed=y> [Source on real world costs of admining PAN-OS]
7. <https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf> [source on how much traffic is encrypted on the web rn]
8. <https://docs.paloaltonetworks.com/pan-os/8-1/pan-os-admin/app-id/app-id-overview> [How App-ID works PAN]
9. https://www.researchgate.net/publication/45601991\_Intrusion\_Detection\_Systems\_-\_Analysis\_and\_Containment\_of\_False\_Positives\_Alerts [Unavoidable High False Positive Rates]
10. <https://www.usenix.org/legacy/event/lisa07/tech/full_papers/bolzoni/bolzoni_html/> [Additional High Alarm Stuff]
11. <https://www.ft.com/content/2fa5bade-cb09-11e8-9fe5-24ad351828ab> [Darktrace False Alarm Stuff]
12. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7181010/> [Masquerade Attacks Pretending to be normal users to avoid AIDS]

<https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-pg-projects/Detecting-Malware-in-TLS-Traf%EF%AC%81c.pdf>

Olivier Roques

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

John Althouse

<https://www.darktrace.com/en/blog/beyond-the-hash-how-unsupervised-machine-learning-unlocks-the-true-power-of-ja-3/>

Max Heinemeyer

<https://www.bc-security.org/post/ja3-s-signatures-and-how-to-avoid-them/>

CX01N

<https://perception-point.io/resources/education/cloud-storage-security-risks/>

Benny Reich

<https://blogs.akamai.com/sitr/2019/05/bots-tampering-with-tls-to-avoid-detection.html>

Moshe Zioni

<https://github.com/salesforce/ja3>