Ryan Boehme, Bryan Nguyen, Wendy Segura

Professor Tim Mather

CIS 55

1 July 2020

**Ray’s Original Network Intrusion Detection Systems:**

**Now in Nacho Cheese and Cool Ranch Flavors**

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 analyzes 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.

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 (Ayub, William and Talbert 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.

It is important to mention a third detection method called stateful protocol analysis that includes similar characteristics of both anomaly and signature-based detections systems. Stateful protocol analysis works by profiling and establishing baseline behavior, and flags deviations to this protocol. By relying on vendor-developed universal standards of normal profiled behavior, this methodology “is capable of understanding and tracking the state of network, transport, and application protocols that have a notion of state” (Latha 5).

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 in (Fig. 1) is a snippet of what the snort.conf rules look like.

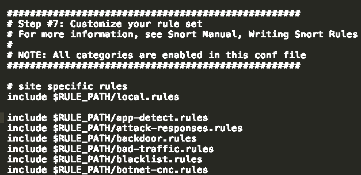


Fig. 1. Snort Configuration File

Roesch, Martin. *Snort 3 User Manual*, Cisco, 2014.

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. It effectively becomes an administrator’s nightmare as you are constantly having to create and revise rules to keep up with the demands of the dynamic ever-changing networks of today’s enterprise. New products are introduced, as well as new attack vectors and strategies. As a result, rules can often be created hastily by IT staff thereby losing a fair amount of Snorts functionality.

But aside from there not being a signature for an attack, or creating enough rules to cover everything, other evasion techniques that can be used against SNORT, or any NIDS tool for that matter, is the ability to exploit the Flowbits feature, Obfuscation techniques: Insertion/Evasion, Fragmentation, Polymorphic Shellcode, Pattern Matching Evasion, and Application Hijacking.

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 (Tran, Aib, Al-Shaer, and Boutaba). An example of flowbit usage within the Snort rules can be seen in Fig. 2.

alert tcp any 143 -> any any (msg:"IMAP login"; content:"OK LOGIN"; flowbits:set,logged\_in; flowbits:noalert;)

alert tcp any any -> any 143 (msg:"IMAP LIST"; content:"LIST"; flowbits:isset,logged\_in;)

Fig. 2. Snort Flowbit Rules

Roesch, Martin. *Snort 3 User Manual*, Cisco, 2014.

The first signature sets the flowbit “logged\_in”, then the second rule will test whether the flowbit has been set on the same TCP connection along with the “LIST” content match. Something to note is the flowbits rule “noalert” will suppress the alert from the first rule and so if triggered it will only signal internally to Snort that a needed precondition for the second signature has been met.

It is the second signature that causes an alert to be generated if it is triggered. Another thing to note is that the flowbit “logged\_in” flowbit is set on traffic returned to the client from the IMAP server versus the “isset” criteria, which tests the “logged\_in” flowbit in the second rule on traffic coming from the client. This illustrates the ability of flowbits to place match criteria on communications emanating from both sides of a connection. Due to flowbits inability to apply across multiple TCP connections it can become exploited. It can only apply within single TCP connections or single UDP conversations (forward and reverse flows). So, it is not possible to set a flowbit on one TCP connection and then test whether this flowbit is set in a completely separate connection(Rash).

Another type of obfuscation attack is 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 (Sumit Siddharth). This type of attack occurs when NIDS is less strict in processing packets than the internal network.

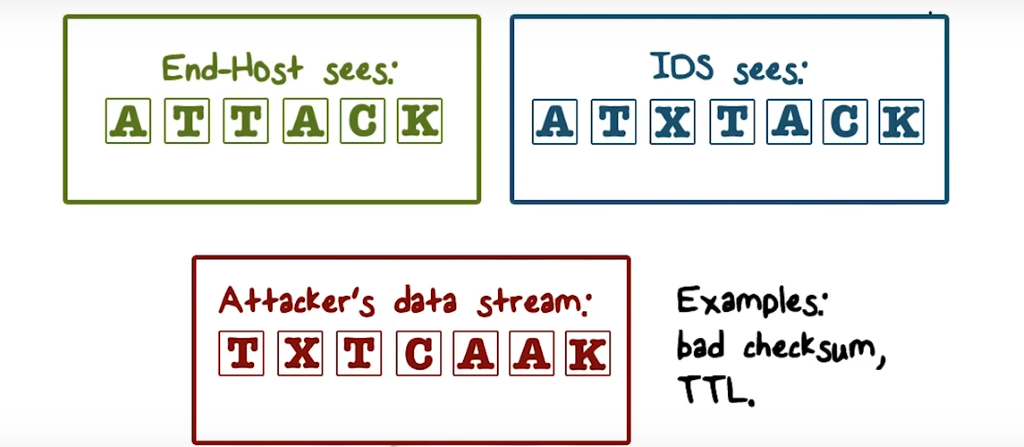


Fig. 3. Snort Rule

Udacity. “*Intro to Information Security*”

*Udacity*, 2016, [www.udacity.com/cours/ud459](http://www.udacity.com/cours/ud459).

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 either 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.

Now we can move onto 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 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 another 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 (Cheng, Lin, Lai, and P.Lin). The illustration below provides an example of how fragmentation might look.

A screenshot of a cell phone

Description automatically generated

Fig. 4. Fragmentation

McAfee. “2019.” *Combating Advanced Evasion Techniques with Network Security Platform*, McAfee, 2019, kc.mcafee.com/corporate/index?page=content&id=KB92003&locale=en\_US.

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.

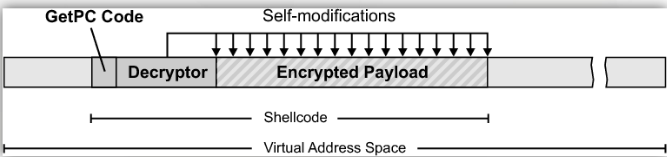


Fig. 5. Polymorphic Shellcode

Damian, Tudor, director. *IDS Evasion Techniques*, DefCamp 2015, 2015, www.youtube.com/watch?v=aQH0DZnJ240.

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. An important note is that IPS do not have the capability to read encryption as it does not have a signature. A possible defense is to set rules to look for high entropy traffic on certain ports in order to catch encrypted traffic. (Cheng, Lin, Lai, and P.Lin).

While one could theoretically spend the time to write an infinite amount of rules to catch all the various permutations that arise due to obfuscation and evasion attacks, the only true solution for catching these attacks is to adopt an “defense in depth” approach. A security analyst must use a variety of tools to catch attacks as they are in transit as well as when they hit their target systems. Additionally, there are tools that scan for vulnerabilities in IDS systems that act as a secondary validation to the signature and rules libraries to ensure proper administration is accounted for. That being said, it still cannot account for zero-day attacks or the extreme resource requirements needed to make an impenetrable signature-based NIDS.

As it became evident that the rigidity of signature-based detection systems could not keep up with the ever-changing landscape of cybersecurity, researchers looked towards new ways of detecting attacks that did not fit a very specific pattern. The idea that attacks could be detected by determining themes and behaviors led to development of anomaly-based systems. While early research showed that these new systems were significantly more flexible in determining new attacks or unrecognized malicious behavior, they also came with a host of their own problems.

lksadfljk;asdfkl;jsdaflk;jsdf


Fig 6. Graphic illustrating basic function of ML algorithm, with acceptable loss values Allemang, Dean. *The Role of a Graph Data Appliance in Machine Learning*, CRAY a Hewelett Packard Enterprise Company, 2013, www.cray.com/blog/role-graph-data-appliance-machine-learning/.

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” (Ayub, William and Talbert 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 (Ayub, William and Talbert 4). 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 (Wiyatno, Xu). In the experiment, using this technique showed a decrease in accuracy of roughly 25% for predicting whether a data sample was an attack (Ayub, William and Talbert 4).

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 (Tramer). 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. (Ayub, William and Talbert 4) 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 (Bolzoni 2). 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 (Victor). 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 (Ram).

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 (Vidal, Monge). 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 a 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 be accepted 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 exploiting SSL encryption, as by definition, encrypted traffic cannot be analyzed. Attackers use this to their advantage by disguising their malicious attacks and thereby completely evading deep packet inspection (Roques 7). 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 7 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.

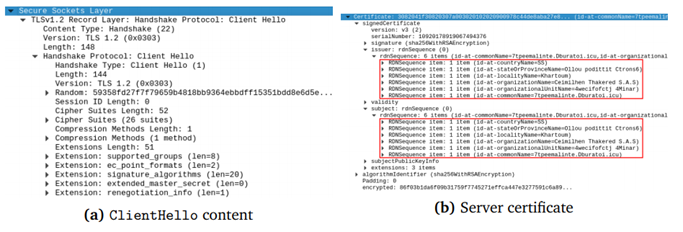


Fig. 7. Client Hello Packet of a Dridex Malware. Note, the only abnormality would be a self-signed certificate. Source: Olivier Roques, *Detecting Malware in TLS Traffic,* 2019.

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, this not only leads to the possibility of false-positives and false-negatives, JA3 fingerprints are also susceptible to tampering.

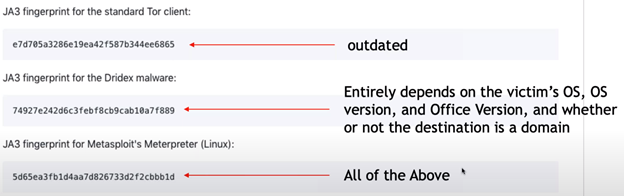


Figure 8: JA3 fingerprints are not consistent. Source: Troy Kent, Dave Shackleford, *(JA) 3 Reasons to Rethink Your Encrypted Traffic Analysis Strategies Webinar,* Awake Security2018*,* https://awakesecurity.com/webinars/ja3-reasons-to-rethink-your-encrypted-traffic-analysis-strategies-webinar/

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 method of signature tampering, attackers utilized bots to 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”). Threat actors can evade detection by tinkering with TLS signatures, even utilizing TLS to masquerade as any other application to get past detection. In conjunction with encryption-based techniques, phishing becomes a more powerful and convincing exploit in fooling the Layer 8 problem, i.e. users.

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.

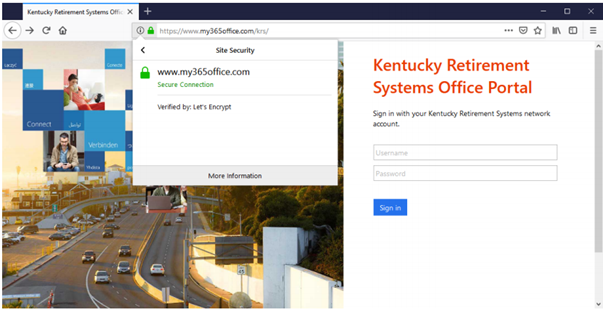


Fig. 9. A convincing phishing website with a green padlock seal of approval.

Source: Karen, *Service Ticket #92833 hacked pls advice,* Redacted Insurance Company, 2020.

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 have proven so successful in remaining among the top threat action varieties (2020 Verizon Data Breach Investigations Report).

As demonstrated, there are many evasive techniques that can be employed to defeat both signature and anomaly-based Network Intrusion Detection Systems. As NIDS become a thing of the past, many new products have risen to take their place. However, they are built upon the shoulders of a flawed system and so inherit those systems flaw. NIDS, and the various systems designed to improve on them, can be a useful tool in a diverse and well vetted toolset to protect network infrastructure, but it should never be relied on as a foolproof defense. 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. A determined attacker, no matter what marketing literature may have you believe, can always find a way around a defense. In an era of increasingly well-organized cybercriminals, it is more important than ever to constantly stay apprised of new developments in the information security field.

Works Cited

1. Roesch, Marty. “Network Intrusion Detection & Prevention System.” *Snort*, Cisco, 2020, www.snort.org/configurations.
2. Siddharth, Sumit. “Endpoint Protection.” *Endpoint Protection - Symantec Enterprise*, Dec. 2005, community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=0e0383b5-1e34-42e0-afaa-65b0cadffd6e.
3. Tran, Tung, et al. “An Evasive Attack on SNORT Flowbits.” *noms2012*, Stonybrook.edu, 2012, www3.cs.stonybrook.edu/~ttran/noms2012.pdf.
4. Lahoti, Laxmi, et al. “Modified Apriori Approach for Evade Network Intrusion Detection System.” *Https://Arxiv.org/Abs/1411.6777*, Cornell University, 2014, arxiv.org/ftp/arxiv/papers/1411/1411.6777.pdf.
5. Abbasi, Ali, et al. “On Emulation-Based Network Intrusion Detection Systems.” *EBNIDS*, RUHR Universitat Bochum, 2017, www.syssec.ruhr-uni-bochum.de/media/emma/veroeffentlichungen/2017/11/09/EBNIDS.pdf.
6. Cheng, Tsung-Huan, et al. “Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems.” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, 2012, pp. 1011–1020., doi:10.1109/surv.2011.092311.00082.
7. Ayub, Ahsan, et al. “2020 54th Annual Conference on Information Sciences and Systems.” Researchgate.net, *Model Evasion Attack on Intrusion Detection Systems Using Adversarial Machine Learning*, 2020, pp. 1–7, www.researchgate.net/publication/340998746\_Model\_Evasion\_Attack\_on\_Intrusion\_Detection\_Systems\_using\_Adversarial\_Machine\_Learning.
8. Allemang, Dean. *The Role of a Graph Data Appliance in Machine Learning*, CRAY a Hewelett Packard Enterprise Company, 2013, www.cray.com/blog/role-graph-data-appliance-machine-learning/.
9. Tramer, Florian, et al. “25th USENIX Security Symposium, USENIX Security 16.” Researchgate.net, *Stealing Machine Learning Models via Prediction APIs*, 2016, pp. 1–20, www.researchgate.net/publication/308027534\_Stealing\_Machine\_Learning\_Models\_via\_Prediction\_APIs.
10. Wiyatno, Rey, and Anqi Xu. “Maximal Jacobian-Based Saliency Map Attack.” *ArXiv*, 2018, arxiv.org/search/?query=Maximal+Jacobian-based+Saliency+Map+Attack.
11. Korhonen, Joni. *Outbound SSL/TLS Decryption Security Impact of SSL/TLS Interception*, 2019, www.theseus.fi/bitstream/handle/10024/226340/Korhonen\_Joni.pdf?sequence=2&isAllowed=y.
12. And Affiliates, Cisco. *Cisco Encrypted Traffic Analytics*, 2019, www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf.
13. ALTO NETWORKS, PALO. *PAN-OS® Administrator’s Guide*, 2020, docs.paloaltonetworks.com/pan-os/8-1/pan-os-admin/app-id/app-id-overview.
14. G. Jacob Victor, et al. “Intrusion Detection Systems - Analysis and Containment of False Positives Alerts .” *International Journal of Computer Applications* , vol. 8, no. 6, 2010, doi:10.5120/931-1308.
15. Bolzoni, Damiano, et al. *ATLANTIDES: An Architecture for Alert Verification in Network Intrusion Detection Systems*, 2007, www.usenix.org/legacy/event/lisa07/tech/full\_papers/bolzoni/bolzoni\_html/.
16. Vidal, Jorge Maestre, and Marco Antonio Sotelo Monge. “Obfuscation of Malicious Behaviors for Thwarting Masquerade Detection Systems Based on Locality Features.” *Sensors*, vol. 20, no. 7, 2020, p. 2084., doi:10.3390/s20072084.
17. Roques, Olivier. *Detecting Malware in TLS Traffic*, 2019, www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-pg-projects/Detecting-Malware-in-TLS-Traf%EF%AC%81c.pdf.
18. Althouse, John. *TLS Fingerprinting with JA3 and JA3S*, 2019, engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.
19. Heinemeyer, Max. “Beyond the Hash: How Unsupervised Machine Learning Unlocks the True Power of JA3.” *Darktrace*, 2018, www.darktrace.com/en/blog/beyond-the-hash-how-unsupervised-machine-learning-unlocks-the-true-power-of-ja-3/.
20. Krasnov, Jacob, and Anthony Rose. “JA3/S Signatures and How to Avoid Them.” *BC Security*, 17 Apr. 2020, www.bc-security.org/post/ja3-s-signatures-and-how-to-avoid-them/.
21. Reich, Benny. “CASB Cloud Security - Risks of Cloud Storage.” *Perception Point*, 27 Aug. 2019, perception-point.io/resources/education/cloud-storage-security-risks/.
22. Threat Research Team May 15, 2019 8:00 AM 0 Comments. “Akamai Security Intelligence & Threat Research Subscribe.” *Bots Tampering with TLS to Avoid Detection - Akamai Security Intelligence and Threat Research Blog*, 2019, blogs.akamai.com/sitr/2019/05/bots-tampering-with-tls-to-avoid-detection.html.
23. Althouse, John, et al. “Salesforce/ja3.” *GitHub*, 2018, github.com/salesforce/ja3.
24. Latha, Keerthi. “Intrusion Detection and Prevention.” *Juniper Networks TechLibrary*, 2016, www.juniper.net/documentation.
25. Townsend , Kevin. “Encrypted Network Traffic Comes at a Cost.” *SecurityWeek*, 2016, www.securityweek.com/encrypted-network-traffic-comes-cost.
26. Ram, Aliya. “Inside Darktrace, the UK's $1.65bn Cyber Security Start-Up.” *Subscribe to Read | Financial Times*, Financial Times, 10 Oct. 2018, www.ft.com/content/2fa5bade-cb09-11e8-9fe5-24ad351828ab.