TCP/IP Data Normalization

Smita Verma
M.Tech. (CSE)
UKTECH, Dehradoon

Garima Krishna
COER, Roorkee
UKTECH, Dehradoon

ABSTRACT

Defending networks against today's attackers is especially challenging for modern intrusion detection/prevention systems for two reasons: the sheer amount of state they must maintain, and the possibility of resource exhaustion attacks on the defense system itself. Our work shows how to cope with these challenges in the context of a TCP stream normalizer whose job is to detect all instances of inconsistent TCP retransmissions.

Keywords

TCP, IP, Data Normalization

1. INTRODUCTION

1.1Normalizer: Network intrusion detection systems' fundamental property is the ability of a skilled attacker to evade detection by exploiting ambiguities in the traffic stream as seen by the monitor. Network intrusion detection and prevention systems are widely used to improve the security of networks used by providers, enterprises, and even home users. etwor They observe all traffic coming in and out of the network and flag or block activities that appear malicious. A normalizer is a network element that prevents evasion attempts by removing ambiguities in network traffic. It sits directly in the path of traffic into a site and observes the packet stream to eliminate potential ambiguities before going into the network.

Normalizer differs from a firewall in several ways. It does not prevent access to services on internal hosts, but ensures that access to these hosts takes place in a secure manner that is unambiguous to the site's Network Intrusion Detection System. Normalizers can prevent known attacks, or shut down access to internal machines from an external host when it detects a probe or an attack. It can shut down and discard state for flows that do not appear to be making progress, while passing and normalizing those that do make progress. In the next section we briefly discuss the possible ways in which a normalizer can be implemented, the various types of normalizers as well as the techniques which are used to develop them.

Various types of Normalizers implemented and related research done in this field

1.1 Recent work done by G. Varghese, J. A. Fingerhut, and F. Bonomi, "Detecting Evasion Attacks at High Speeds without Reassembly," addresses one type of evasions, namely an attacker attempting to prevent a specific signature match against text they transmit. The authors developed a scheme based on introducing a modest change in end-system TCP behavior in order to allow a monitor to detect attempts to ambiguously transmit byte sequences that match a given set of signatures. Their scheme is appealing in that by exploiting the introduced end-system change, they avoid needing to reassemble TCP byte streams. However, their scheme is also significantly limited in that it only applies to evasions that correspond to directly manipulating a known byte-sequence signature. As such, the scheme does not handle cases where the ambiguity does not constitute an actual attack in itself, but only confuses the monitor's protocol parsing and obscures the occurrence of an attack later in the stream. [8.2]

1.1Y. Sugawara, M. Inaba, and K. Hiraki in their paper "High-speed and Memory Efficient TCP Stream Scanning Using FPGA," describe an FPGA-based solution to efficient TCP stream-level signature detection. Their system detects inconsistent retransmissions by storing hashes of transmitted packets. To handle retransmissions that do not overlap with original segment boundaries, the authors simply propose holding onto the partial overlaps till other packets that "fill the gap" arrive. However, our trace evaluation shows that such an approach will result in a significant number of connections stalling on pending consistency checks; RoboNorm addresses this problem with the ACK promotion mechanism.[8.3]

1.2 Normalization as a general feature has been incorporated into secure operating systems and commercial products. Some of these latter include explicit options to check for inconsistent retransmissions, but do not provide technical details as to how such detection works. From informal discussions with other vendors, it appears that a common approach is to use payload hashes, but without addressing the crucial problem of misaligned retransmissions for which the hashes cannot be matched.

1.3 Shankar and Paxson explored a different approach to defending against evasion attacks which they term "Active Mapping". Here, the idea is for the network monitor to proactively determine how specific end systems and network paths will resolve potential ambiguities. While this approach is a valid point in the overall design space, we argue that eliminating ambiguities rather than attempting to correctly guess their outcome, provides a more robust foundation for security monitoring technology.

1.4 Work by Levchenko et al. demonstrates in formal terms that many security detection tasks (e.g., detecting SYN flooding, port scans, connection hijacking and evasion attacks) fundamentally require maintaining per-connection state. This finding highlights the importance of reducing the amount of per-connection state.

1.5 In work done by S. Dharmapurikar and V. Paxson, "Robust TCP Stream Reassembly in the Presence of Adversaries," explores how to robustly reassemble TCP byte streams when faced with adversaries who attempt to overwhelm the accompanying state management. Reassembly involves maintaining out-of-order data only until sequence "holes" are filled, while normalization requires maintaining data until it is acknowledged and hence requires a different solution.[8.4]

2. LITERATURE REVIEW

2.1 Evasion Attacks: The reviewed literature presents a keen insight into another important class of network security attacks i.e. the evasion attacks. Evasion is a term used to describe techniques of bypassing an information security device in order to deliver an exploit, attack or other malware to a target network or system, without detection.[8.1] Evasions are typically used to counter network-based intrusion detection and prevention systems (IPS, IDS) but can also be used to by-pass firewalls. A further target of evasions can be to crash a network security device, rendering it in-effective to subsequent targeted attacks. Evasions can be particularly nasty because a well-planned and implemented evasion can enable full sessions to be carried forth in packets that evade IDS. Attacks carried in such sessions will happen right under the nose of the network and service administrators. The security systems are rendered ineffective against well-designed evasion techniques, in the same way a stealth fighter can attack without detection by radar and other defensive systems. Network attackers often use network IPS evasion techniques to attempt to bypass the intrusion detection, prevention, and traffic filtering functions provided by network IPS sensors. [8.6]Some commonly used network IPS evasion techniques are listed below:

- **2.1.1** Encryption and Tunneling
- **2.1.2** Timing Attacks
- **2.1.3** Resource Exhaustion
- **2.1.4** Traffic Fragmentation
- **2.1.5** Protocol-level Misinterpretation
- **2.1.6** Traffic Substitution and Insertion

2.1.1 Encryption and Tunneling: One common method of evasion used by attackers is to avoid detection simply by encrypting the packets or putting them in a secure tunnel. As discussed now several times, IPS sensors monitor the network and capture the packets as they traverse the network, but network based sensors rely on the data being transmitted in plaintext. When and if the packets are encrypted, the sensor captures the data but is unable to decrypt it and cannot perform meaningful analysis. This is assuming the attacker has already established a secure session with the target network or host. Some examples that can be used for this method of encryption and tunneling are:

- **2.1.1.1** Secure Shell (SSH) connection to an SSH server
- **2.1.1.2** Client-to-LAN IPSec (IP Security) VPN (virtual private network) tunnel
- **2.1..1.3** Site-to-site IPSec VPN tunnel
- **2.1.1.4** SSL (Secure Socket Layer) connection to a secure website

There are other types of encapsulation that the sensor cannot analyze and unpack that attackers often use in an evasion attack. For example, GRE (Generic Route Encapsulation) tunnels are often used with or without encryption.

2.1.2Timing Attacks: Attackers can evade detection by performing their actions slower than normal, not exceeding the thresholds inside the time windows the signatures use to correlate different packets together. These evasion attacks can be mounted against any correlating engine that uses a fixed time window and a threshold to classify multiple packets into a composite event. An example of this type of attack would be a very slow reconnaissance attack sending packets at the interval of a couple per minute. In this scenario, the attacker would likely evade detection simply by making the scan possibly unacceptably long.

2.1.3Resource Exhaustion: A common method of evasion used by attackers is extreme resource consumption, though this subtle method doesn't matter if such a denial is against the device or the personnel managing the device. Specialized tools can be used to create a large number of alarms that consume the resources of the IPS device and prevent attacks from being logged. These attacks can overwhelm what is known as the management systems or server, database server, or out-of-band (OOB) network. Attacks of this nature can also succeed if they only overwhelm the administrative staff, which does not have the time or skill necessary to investigate the numerous false alarms that have been triggered. Intrusion detection and prevention systems rely on their ability to capture packets off the wire and analyze them quickly, but this requires the sensor has adequate memory capacity and processor speed. The attacker can cause an attack to go undetected through the process of flooding the network with noise traffic and causing the sensor to capture unnecessary packets. If the attack is detected, the sensor resources may be exhausted but unable to respond within a timely manner due to resources being exhausted.[8.7]

2.1.4Traffic Fragmentation: Fragmentation of traffic was one of the early network IPS evasion techniques used to attempt to bypass the network IPS sensor. Any evasion attempt where the attacker splits malicious traffic to avoid detection or filtering is considered a fragmentation-based evasion by:

- **2.1.4.1** Bypassing the network IPS sensor if it does not perform any reassembly at all.
- **2.1.4.2** Reordering split data if the network IPS sensor does not correctly order it in the reassembly process.
- **2.1.4.3** Confusing the network IPS sensor's reassembly methods which may not reassemble split data correctly and result in missing the malicious payload associated with it.
- **2.1.4.4** A few classic examples of fragmentation-based evasion are below:
- **2.1.4.5** TCP segmentation and reordering, where the sensor must correctly reassemble the entire TCP session, including possible corner cases, such as selective ACKs and selective retransmission.
- **2.1.4.6** IP fragmentation, where the attacker fragments all traffic if the network IPS does not perform reassembly. Most sensors do perform reassembly, so the attacker fragments the IP traffic in a manner that it is not uniquely interpreted. This action causes the sensor to interpret it differently from the target, which leads to the target being compromised.

In the same class of fragmentation attacks, there is a class of attacks involving overlapping fragments. In overlapping fragments the offset values in the IP header don't match up as they should, thus one fragment overlaps another. The IPS sensor may not know how the target system will reassemble these packets, and typically different operating systems handle this situation differently.[8.2]

2.1.5 Protocol-level Misinterpretation: Attackers also evade detection by causing the network IPS sensor to misinterpret the end-to-end meaning of network protocols. In this scenario the traffic is seen differently from the target by the attacker causing the sensor either to ignore traffic that should not be ignored or vice versa. Two common examples are packets with bad TCP checksum and IP TTL (Time-to-live) attacks.[8.5]

A bad TCP checksum could occur in the following manner: An attack intentionally corrupts the TCP checksum of specific packets, thus confusing the state of the network IPS sensor that does not validate checksums. The attacker can also send a good payload with the bad checksum. The sensor can process it, but most hosts will not. The attacker follows with a bad payload with a good checksum. From the network IPS sensor this appears to be a duplicate and will ignore it, but the end host will now process the malicious payload.[8.3]

The IP TTL field in packets presents a problem to network IPS sensor because there is no easy way to know the number of hops from the sensor to the end point of an IP session stream. Attackers can take advantage of this through a method of reconnaissance by sending a packet that has a very short TTL which will pass through the network IPS fine, but be dropped by a router between the sensor and the target host due to a TTL equaling zero. The attacker may then follow by sending a malicious packet with a long TTL, which will make it to the end host or target. The packet looks like a retransmission or duplicate packet from the attacker, but to the host or target this is the first packet that actually reached it. The result is a compromised host and the network IPS sensor ignored or missed the attack.

2.1.6 Traffic Substitution and Insertion: Another class of evasion attacks includes traffic substitution and insertion. Traffic substitution is when that attacker attempts to substitute payload data with other data in a different format, but the same meaning. A network IPS sensor may miss such malicious payloads if it looks for data in a particular format and doesn't recognize the true meaning of the data. Some examples of substitution attacks are below

1 Substitution of spaces with tabs, and vice versa, for example inside HTTP requests.
2. Using Unicode instead of ASCII strings and characters inside HTTP requests.
3 Exploit mutation, where specific malicious shell code (executable exploit code that forces the target system to execute it) can be substituted by completely different shell code with the same meaning and thus consequences on the end host or target.
4 Exploit case sensitivity and changing case of characters in a malicious payload, if the network IPS sensor is configured with case-sensitive signature.

Insertion attacks act in the same manner in that the attacker inserts additional information that does not change the payload meaning into the attack payload. An example would be the insertion of spaces or tabs into protocols that ignore such sequences.[8.4]

Intrusion detection is an important component of a security system, and it complements other security technologies. By providing information to site administration, ID allows not only for the detection of attacks explicitly addressed by other security components (such as firewalls and service wrappers), but also attempts to provide notification of new attacks unforeseen by other components. Intrusion detection systems also provide forensic information that potentially allow organizations to discover the origins of an attack. In this manner, ID systems attempt to make attackers more accountable for their actions, and, to some extent, act as a deterrent to future attacks.[8.8]

Given the implications of the failure of an ID component, it is reasonable to assume that ID systems are themselves logical targets for attack. A smart intruder who realizes that an IDS has been deployed on a network she is attacking will likely attack the IDS first, disabling it or forcing it to provide false information (distracting security personnel from the actual attack in progress, or framing someone else for the attack).

In order for a software component to resist attack, it must be designed and implemented with an understanding of the specific means by which it can be attacked. Unfortunately, very little information is publicly available to IDS designers to document the traps and pitfalls of implementing such a system. Furthermore, the majority of commercially available ID systems have proprietary, secret designs, and are not available with source code. This makes independent third-party analysis of such software for security problems difficult.

The most obvious aspect of an IDS to attack is its ``accuracy". The ``accuracy" of an IDS is compromised when something occurs that causes the system to incorrectly identify an intrusion when none has occurred (a ``false positive" output), or when something occurs that causes the IDS to incorrectly fail to identify an intrusion when one has in fact occurred (a ``false negative"). Some researchers discuss IDS failures in terms of deficiencies in ``accuracy" and ``completeness", where ``accuracy" reflects the number of false positives and ``completeness" reflects the number of false negatives.

Other attacks might seek to disable the entire system, preventing it from functioning effectively at all. We say that these attacks attempt to compromise the ``availability" of the system.

3. OBJECTIVE

The objective of this project is to implement a network element that detects and blocks inconsistent retransmissions in any TCP byte stream, in a manner that takes care of the memory requirements and is also resistant to attacks.

The programs will run on Linux operating system and the entire project will be developed and coded using C language and socket programming.

4. SCOPE OF THE PROJECT

We have envisaged our normalizer to work on a Linux platform on a three node network with a simple client-server architecture. Our endeavor is to implement the functionality of this essential network element, the normalizer in software using the C programming language. The same can be extended to hardware implementation with the aid of assembly language. The scope of this normalizer can be extended to other dynamic network configurations and on other platforms (Windows, Mac etc.). There are still possibilities of exploring a different approach to defending against evasion attacks in which the network monitor may proactively determine how specific end systems and network paths will resolve potential ambiguities. While this approach can be a valid point in the overall design space, eliminating ambiguities, rather than attempting to correctly guess their outcome, seems to provide a more robust foundation for security monitoring technology

5 RESULT AND DISCUSSION

Defending networks against today's attackers is especially challenging for modern intrusion detection/prevention systems for two reasons: the sheer amount of state they must maintain, and the possibility of resource exhaustion attacks on the defense system itself. Our work shows how to cope with these challenges in the context of a TCP stream normalizer whose job is to detect all instances of inconsistent TCP retransmissions. The two currently used methods to detect inconsistent retransmissions, maintaining complete contents of unacknowledged data, or maintaining only the corresponding hashes suffer from a set of flaws each. Systems that maintain complete contents consume an amount of memory problematic for high-speed operation. Systems that maintain hashes cannot verify the consistency of the 20-30% of retransmissions that fail to preserve original segment boundaries; as a result attackers can easily encode their evasions in these unverified segments. Our normalizer stores hashes of data and verifies the consistency of all retransmissions. The resulting design is necessarily somewhat complex. In considering resource exhaustion attacks, the observation that provisioning for a worst-case traffic pattern is simply impractical led us to develop a simple framework to evict connections when space is at a premium. Thus, our most important conclusion is that TCP stream normalization does not have to choose between correctness and implement ability; it can achieve both goals, while resisting a range of resource exhaustion attacks. **[8.9]**

6 . LIMITATIONS

We have implemented the normalizer for TCP at user-level. For high performance a production normalizer would need to run in the kernel rather than at user level, but our current implementation makes testing, debugging and evaluation much simpler.

The application of this project seems more probable on a high speed network where the internal network of a home user/ organization needs to be protected from any malicious activity by an attacker. That would require full control on the transmission of packets so that the inconsistent retransmissions can be blocked and not allowed to pass by the normalizer.

7 . CONCLUSION

The work done in this semester brings us to the end of our Project. Having prepared the three key components namely, the packet sniffer program, the SHA -1 implementation and the setting up of a client server network through socket programming, we were able to combine them to prepare our TCP stream normalizer. These three components essentially form the major part of the requirements for implementing a robust and efficient network normalizer. Next came the logical implementation of how the normalizer works to prevent the typical kind of attack that we discussed i.e. the evasion attack. The client server configuration served the purpose of simulating the actual working of a hardware implementation of the normalizer where the normalizer helps prevent our internal network from any inconsistent retransmissions. The packet sniffer allowed the capturing of essential details required to track and distinguish genuine packets from the others. With the help of the SHA-1 hashing algorithm, the memory requirements of our normalizer were reduced great deal. Comparing the incoming packets with the hash codes of previously monitored packets enabled the working of the normalizer that prepares a table for each incoming packet and holds three key entities – the sequence number, identification field and the hash of data payload. The normalizer was thus successful in demarcating genuine packets forwarded by the TCP protocol from any malicious packet that an attacker may have introduced in the TCP stream.

8. REFERENCE

[1] M. Handley,V. Paxson, and C. Kreibich, “Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics,” in *Proc. USENIX Security Symposium, Aug. 2001.*

[2] G. Varghese, J. A. Fingerhut, and F. Bonomi, “Detecting Evasion Attacks at High Speeds without Reassembly,” in *Proc. ACM SIGCOMM, Sept. 2006.*

[3] “Configuring TCP Normalization,” 2006, http://www.cisco.com/en/US/products/ps6120/products configuration guide chapter09186a008054ecb8. html#wp1051891.

[4] Mythili Vutukuru, H. Balakrishnan, “Efficient and Robust TCP Stream Normalization”, IEEE Symposium on Security and Privacy, May 2008.

[5] Anderson, Ross (2008). “Security Engineering – A Guide to Building Dependable Distributed Systems – 2nd edition. John Wiley & Sons.

[6] Burns, David (2011). “CCNP Security IPS 642-627 Official Cert Guide”. Cisco Press.

[7] Thomas H. Ptacek, Timothy N. Newsham, “Insertion, Evasion and denial of service: Eluding Network Intrusion Detection”. Jan 1998.

[8] Tanenbaum, Andrew S., “Computer Networks” 4th edition. Pearson Education. “Secure Hash Standard”, FIPS – 180, National Institute of Standards and Technology.