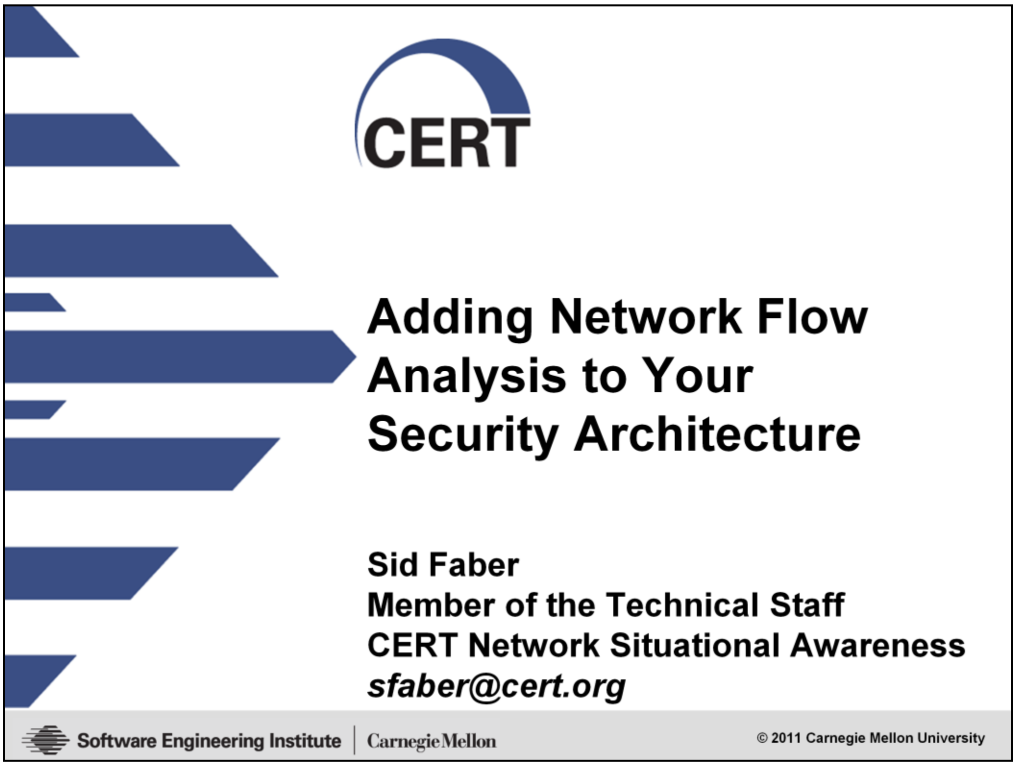
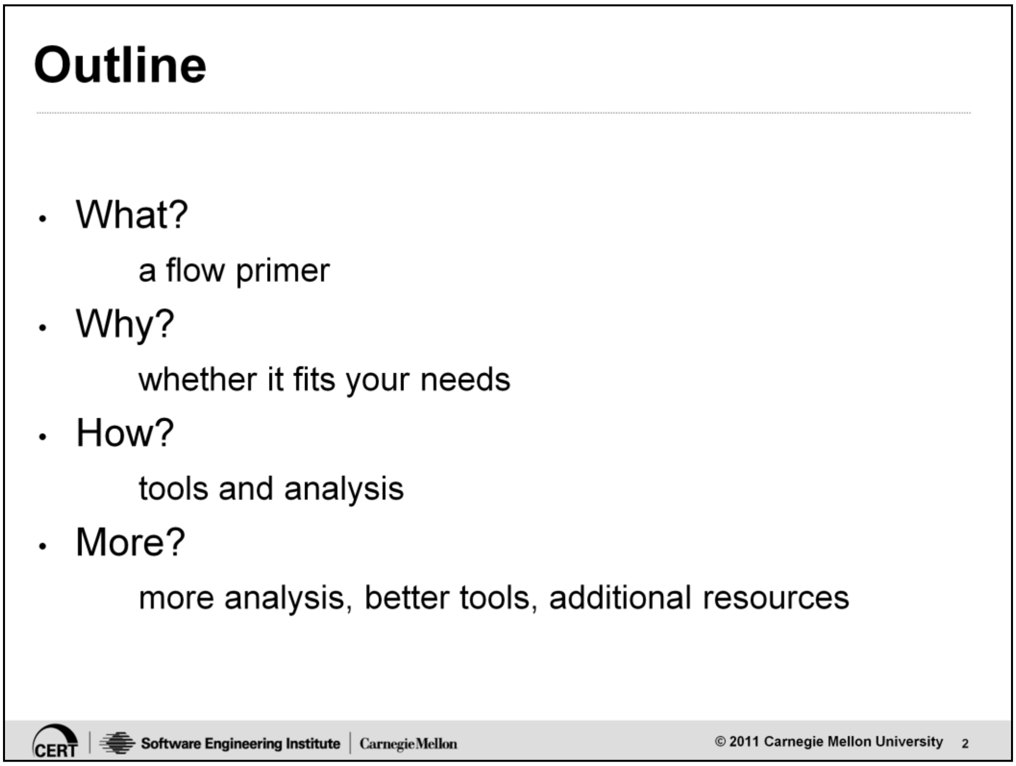


Welcome to this session on Network Flow Analysis.

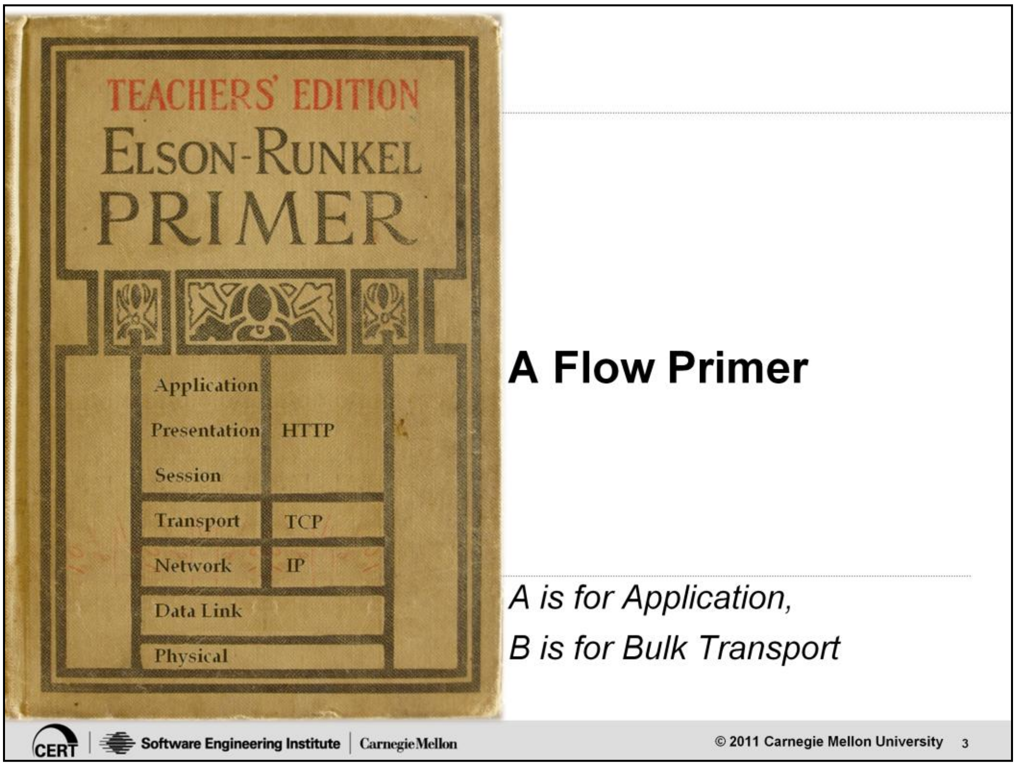


In this session we'll begin by covering the basics of network flow analysis. After a brief look at the history of flow, we'll dig into some examples of how common network traffic appears in flow.

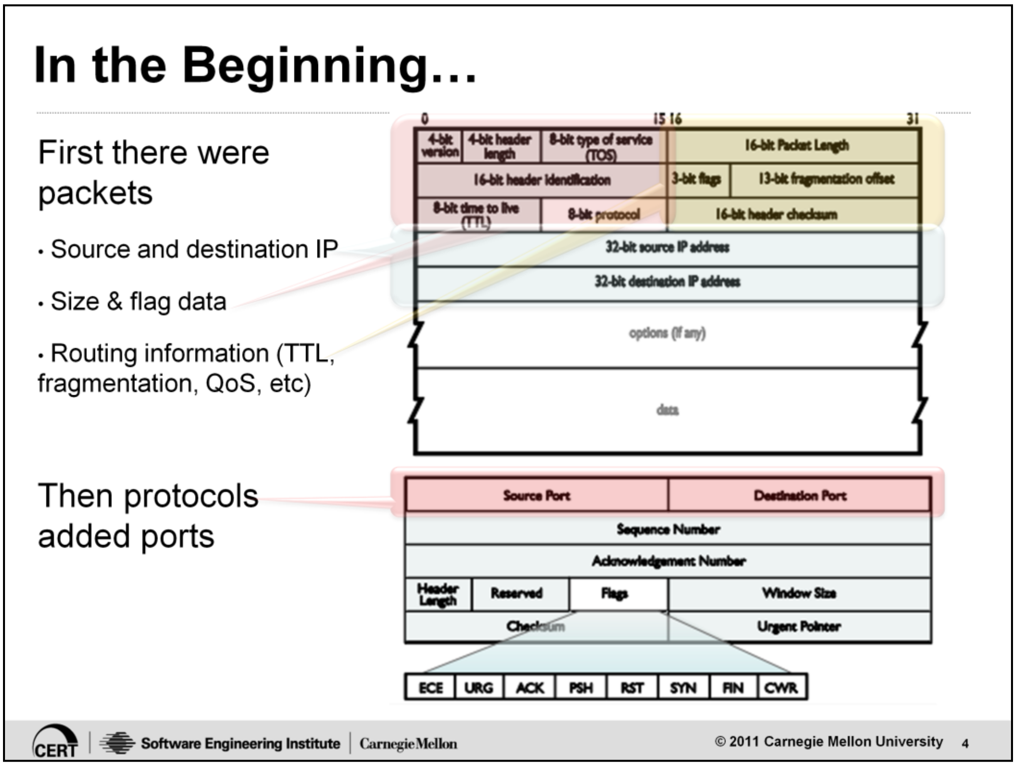
Once the basics have been covered, we'll plug flow into a typical network security plan. There are some places gaps left by other security tools that can be filled by flow, so by the end of this discussion you should be able to decide whether flow might help solve some of your challenge problems.

Now that you've decided to add flow into your architecture, we need to find the right tools. In this section we'll talk about the architecture of the typical SiLK installation.

We'll close with a look at some of the more advanced tools and things that are in the works.



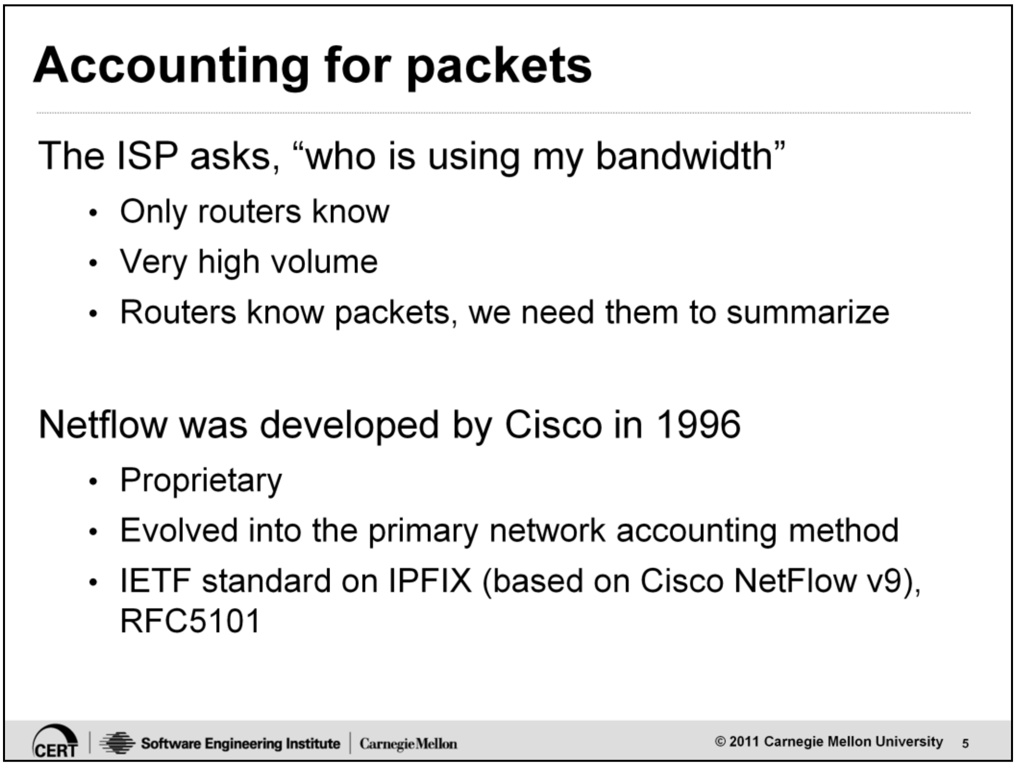
So let’s get started looking at what flow is all about.



Let's take a closer look at the relationship between data packets and flow records.

Here you see the IP and TCP packet header. As you already know, each packet contains the source and destination IP addresses. The IP packet also contains information about packet size, IP flags which help the client interpret the data contained in the packet.

Routing information, and other packet related data in the IP header help get the packet to its destination. TCP and UDP protocols encapsulated within IP packets contain source and destination ports which help the packet find the right service on the destination machine.



So then how did packets end up generating flow? In the early days of Internet routing, the service providers quickly realized that it was very important to know who was using network resources.

Some features of this problem quickly became apparent:

- Only routers tracked the information that was provided,
- The volume of traffic was very high so data collection had to match the volume, and
- Routers are very good at understanding and moving packets, so they can summarize packet data—but they can not summarize actual content data.

As a result, Cisco developed a proprietary standard for netflow data collection. In 1996 that was used for network accounting and network traffic analysis. This standard was quickly adopted by all major routers, and became formalized into an Internet Engineering Task Force Standard based on the most recent version of netflow. The proprietary “netflow” protocol has now become a standard wire-format network “flow” protocol known as IPFIX (Internet Protocol Flow Information Exchange).

What is a Flow?

A flow is an aggregated record of packets

SiLK flows are defined by five unique attributes

- IP Protocol (TCP, UDP, ICMP, IPSec, etc)
- Source Address
- Destination Address
- Source Port
- Destination Port

SiLK flows are unidirectional

CERT | Software Engineering Institute | Carnegie Mellon

 6

As we have already discussed, the flow record is an aggregated summary of packets observed by the flow sensor.

Each flow record combines all the packets that match the five unique packet attributes: IP protocol, source and destination address and source and destination port.

SiLK flows are unidirectional; that is, packets from a client to a server are **not** combined with the server to client packets because the source and destinations are swapped, so typical TCP sessions result in two flow records—one from client to server and another from server to client.

What Does a Flow Know?

Each unique flow (tuple) has associated attributes

- Timing (start, stop)
- Volume (packets, bytes)
- TCP flags
- Collection location (sensor, traffic type)

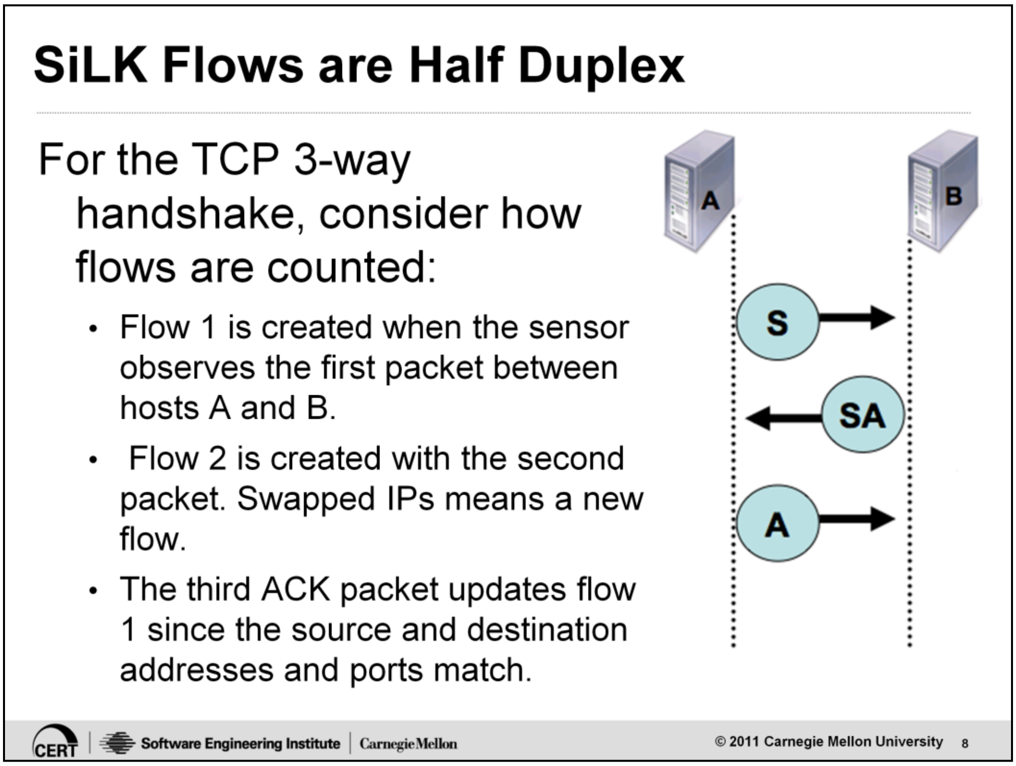
Flows get flushed when they close

- Timeouts, TCP FIN/RST, router resources low

As individual packets are combined into flow records, specific attributes of the packets are recorded and saved with the flow record.

The flow records the time the first packet in the flow was seen, as well as the time the last packet was seen. Remember, individual packets are instantaneous and don’t have time stamps or durations. A flow record counts the total number of packets combined into the record, and the total byte volume for all the packets that make up the flow.

A list of all the TCP flags seen in the flow is saved. The flow record also stores the location for where the flow was collected. Also, remember that flow records can not stay in memory forever, and they must be flushed to the collector. This happens when TCP sessions close and when the flows timeout.



Let’s take another look at the concept of unidirectional flow records.

Consider the TCP three-way handshake, where the client sends a SYN packet to the server, the server responds with a SYN-ACK packet, and the client finishes the handshake with an ACK packet.

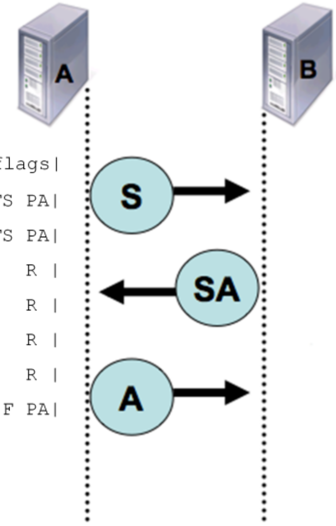
As the first SYN packet passes the sensor, our first flow record is created. The tuple includes the protocol (TCP), the source address A and the destination address B, and has the “flags” value set to SYN only.

B responds with a SYN-ACK packet. This creates a _new_ flow record where the tuple has the source address B and the destination address A.

The third packet, the ACK packet, matches the first flow record, since it has source address A and destination B. This flow is updated with a packet count of two, the byte volume is increased, and the “flags” value is changed from “SYN only” to “SYN plus ACK”. From here, you should be able to judge how additional packets in this TCP session will be added to these two existing flow records.

What is this #1

```
         sIP|          dIP|sPort|dPort|pkt|bytes|flags|
63.236.206.174|   72.24.144.5|44800|   25| 21|19606|FS PA|
   72.24.144.5|63.236.206.174|   25|44800| 17| 1066|FS PA|
63.236.206.174|   72.24.144.5|44800|   25|  1|   40|   R |
63.236.206.174|   72.24.144.5|44800|   25|  1|   40|   R |
63.236.206.174|   72.24.144.5|44800|   25|  1|   40|   R |
63.236.206.174|  72.24.146.90|44800|   25|  1|   40|   R |
  72.24.146.90|63.236.206.174|   25|44800|  1|   49| F PA|
```



At this point we will begin looking at some sample flow data. What is shown in this example? Can you tell which IP address appears to be the client and which is the server?

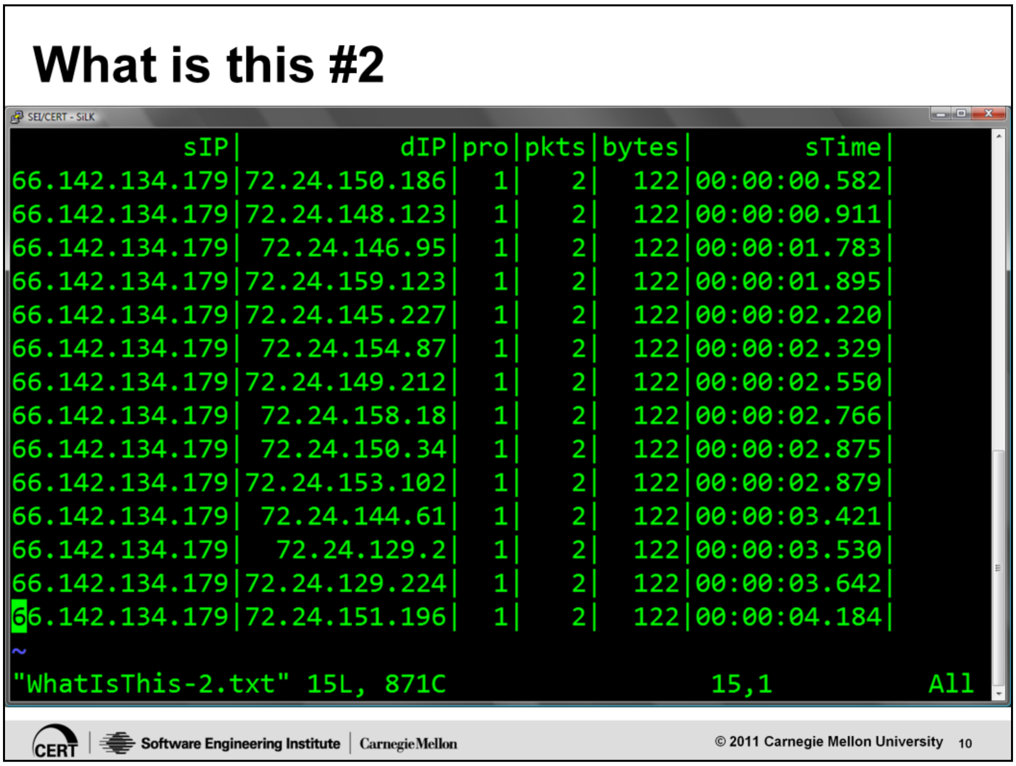
We see communications on port 25, which means this is probably email (SMTP) traffic. IP address 72.24.177.5 appears to be the server since it is hosting the SMTP port 25 service.

Since IP address 63.236.206.174 is using a high numbered ephemeral port, it is the SMTP client.

Take a look at the byte volume. There's a big difference between the client and the server—the client set about five times as much data as the server, a total of about 20k bytes. That makes sense if we consider 63.236.206.174 to be sending a message to 72.24.144.5.

What about all those RST packets? Well, they're all sent from the client to the server, and it seems like they're sent after the FIN connection teardown has completed. Although we can't tell exactly what is going on here, it's not uncommon to see spurious RST packets at the end of a TCP session, particularly with high-volume clients and servers.

Finally, did you notice that there are some extra records here that don't really belong? The last two records may have escaped your attention. They have the same client address and port, and they're SMTP traffic, but they actually point to a different SMTP server.



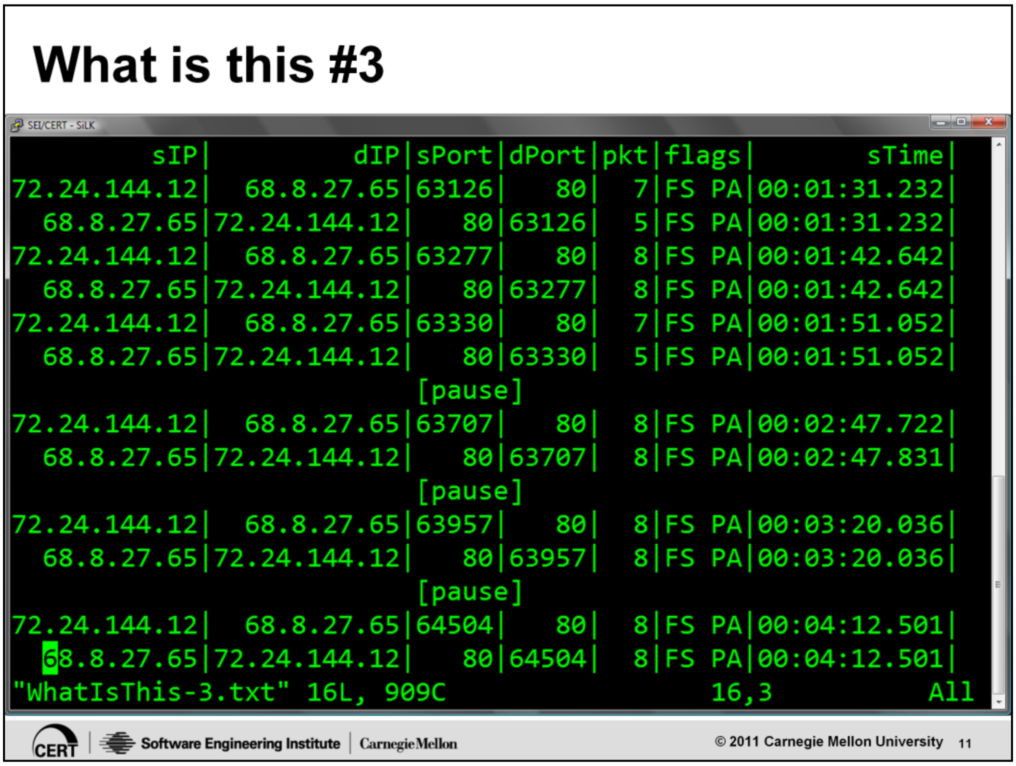
What do you notice in this set of flow records? Does this look like normal traffic to you?

Taking a quick look at the IP addresses, you see that one address—the source address—remains the same, while the destination address changes. In fact, you'll notice that while the destination address changes, it only changes within a given range of addresses; the first two octets of the destination address remain the same while the second to octets change. It looks like the IP address 66.142.134.179 is scanning the 72.24 network. This is a scan.

Can you tell anything else about the scan? Let's take a closer look. The protocol is always 1. That means this is an ICMP scan. Each scan target receives two packets for a total of 122 bytes. That means each packet is probably half that, or 61 bytes. Each packet probably has a 40-byte IP header and four bytes of ICMP header, leaving room for 17 bytes of data.

Finally, look at the packet timing. We see about two to four packets per second. Is this a fast scan, or maybe even a denial-of-service attack? Not at that rate.

Considering everything we've observed, this just looks like routine scan activity.



Next example. At first glance, this seems a bit more confusing than our last example. Is it normal traffic, or something to be concerned about?

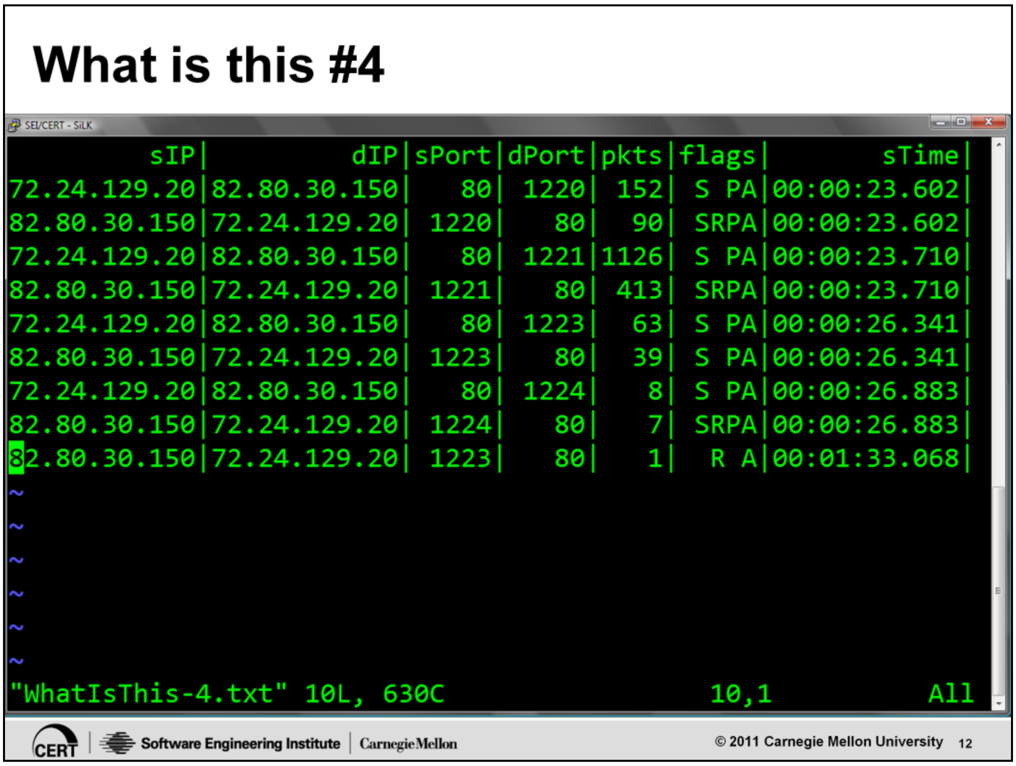
Our first clue in unraveling this traffic is to look at the ports. You should quickly recognize port 80, the port used for normal web or HTTP traffic. Every flow record on the list uses this port, so it's probably all HTTP traffic. Looking at the first record, since 80—the service port—is the destination port, that means that the destination IP address is the server. So we know that 68.8.27.65 is service HTTP traffic.

Similarly, looking at the high port, the ephemeral port, we identify the client as 72.24.144.12. Also notice that the ephemeral port increases with new connections, which is to be expected.

We see three separate connections from the client to the web server all within 20 seconds, and then three later connections. Each connection has normal flags—SYN, ACK and FIN—and a reasonable number of packets. All in all, the first set of six flows—three TCP connections—looks like a standard web page loading and then pulling down some included reference data like images or style sheets.

Finally, take a look at all the flows together. Everything's pretty similar, except for the timing. There's repetitive behavior here, but it's pretty slow—there's almost a minute between each connection—and the timing is not exact. This does not look like system behavior; instead, it looks like normal user interaction with a web site: click, read, click, read, click.

Overall, this looks like normal web browsing activity.

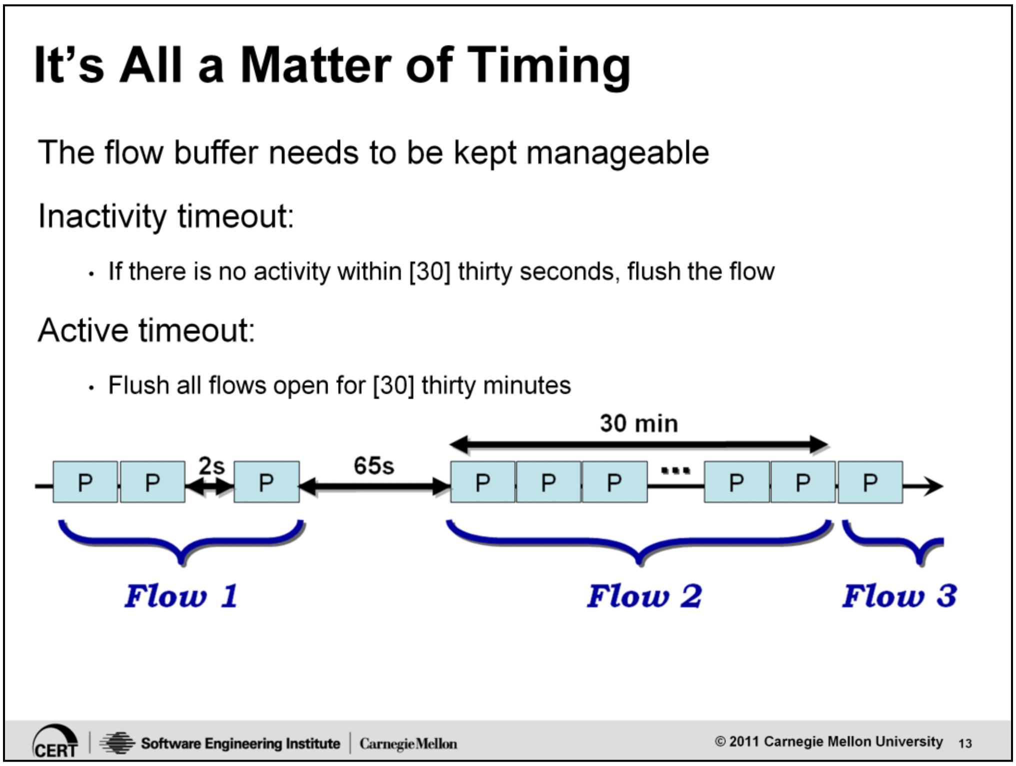


Here's another example. Once again, you see the port 80 traffic right away, and can assume that this is HTTP web browsing traffic. But there's something a bit different about this set of flows. What do you see?

You probably noticed right away that these flows are larger than the ones in the previous example. Even though these sessions are large, this is still fairly normal for web traffic. This could come from large images, or another rich media type.

However, did you notice the RST packets in these flows? In this case, we see resets being sent from the client back to the server, which is not uncommon—the client probably no longer wants the content being delivered. In fact, it's also very common for high-volume web servers to send resets to clients as well, because it's much quicker and more efficient to shut down a TCP session with a RST packet than with a FIN exchange.

Finally, note how the last flow happens long after the actual data exchange in the TCP session. When the client timed out the TCP session it sent this reset packet. In cases like this you have to watch the ephemeral port to make sure you don't accidentally tie the last RST packet in with the wrong TCP session.



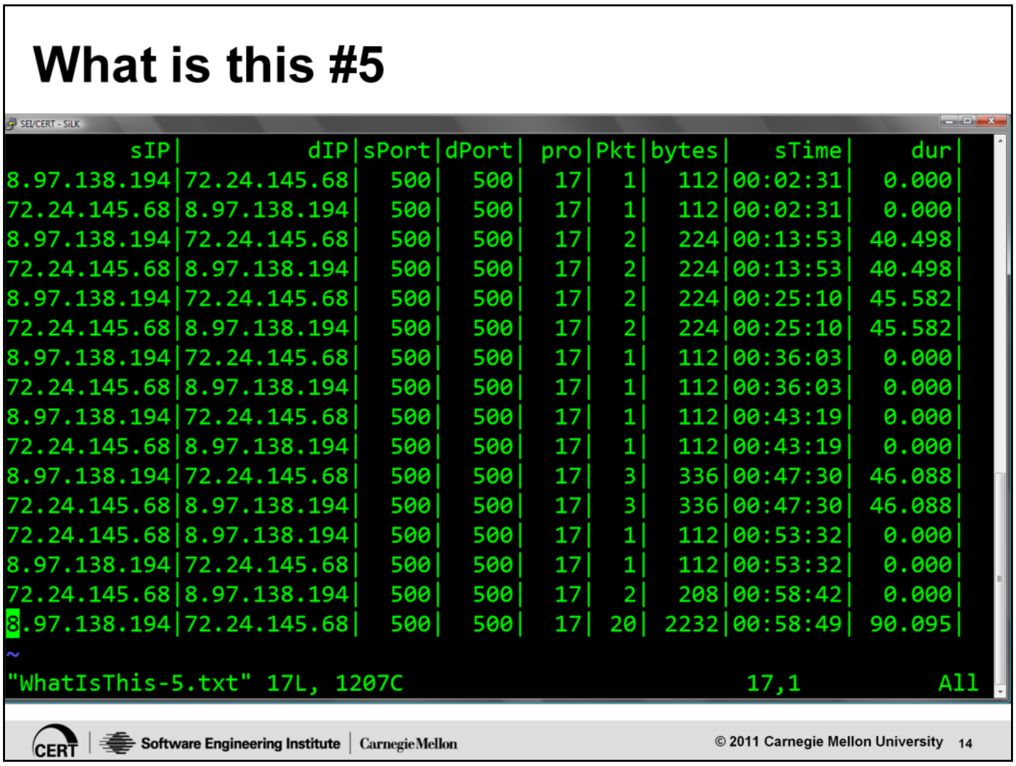
We briefly mentioned that flows have to time out to keep the flow buffer manageable. Let's take a closer look at what causes a flow to time out.

Keep in mind that a "flow" means the tuple—source and destination ports and addresses. An inactive timeout occurs when no packets are seen within the inactive window that match the tuple. Active timeouts occur when a flow has been open for a specific period; this ensures that long-lived flows are flushed off the sensor within a reasonable time.

Taking a look at this activity in a timeline, here you see individual packets with the same tuple, packets that are aggregated into a single flow. Since inter-packet arrival is relatively quick, the packets aggregate nicely. Then at some point, communications on the socket pauses for a period, which causes the flow to time out. This is an inactivity timeout.

After the pause, the conversation becomes active again, and data continues to flow for a long time. At a certain point—30 minutes in this case—the flow collector flushes the flow and starts a new flow. This is an active timeout.

Now let's take a look at flow records that have timeouts.

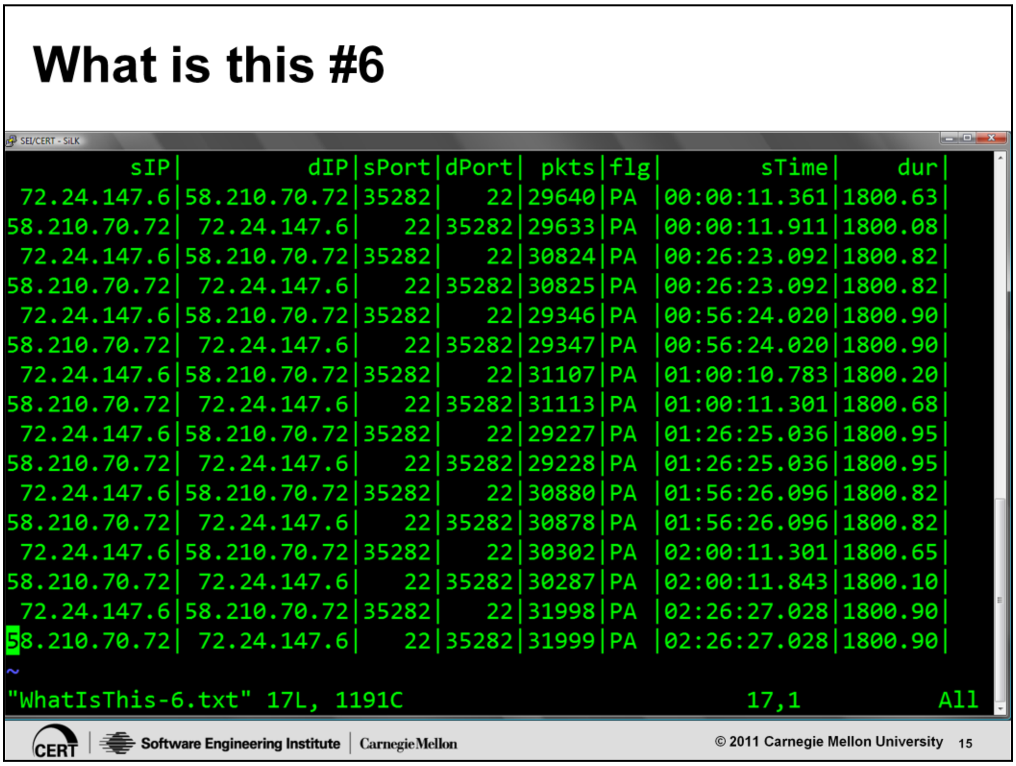


Take a look at this flow data, see if you can tell how flow timeouts affect what you see. Do you recognize this traffic?

It might seem odd that both the source and destination ports are the same for this protocol 17 (UDP) traffic. However, that's actually typical for VPN clients.

Look at the packet and byte counts, however. They're pretty small. Also look at the timing of these packets—there's a lot of time in between each one. What do you think this is?

To me this looks like a mostly quiet VPN, the only thing being sent on it are small keep-alive type packets. That would also explain why the last flow is larger than the rest—at some point, a small amount of data actually traveled across the VPN.



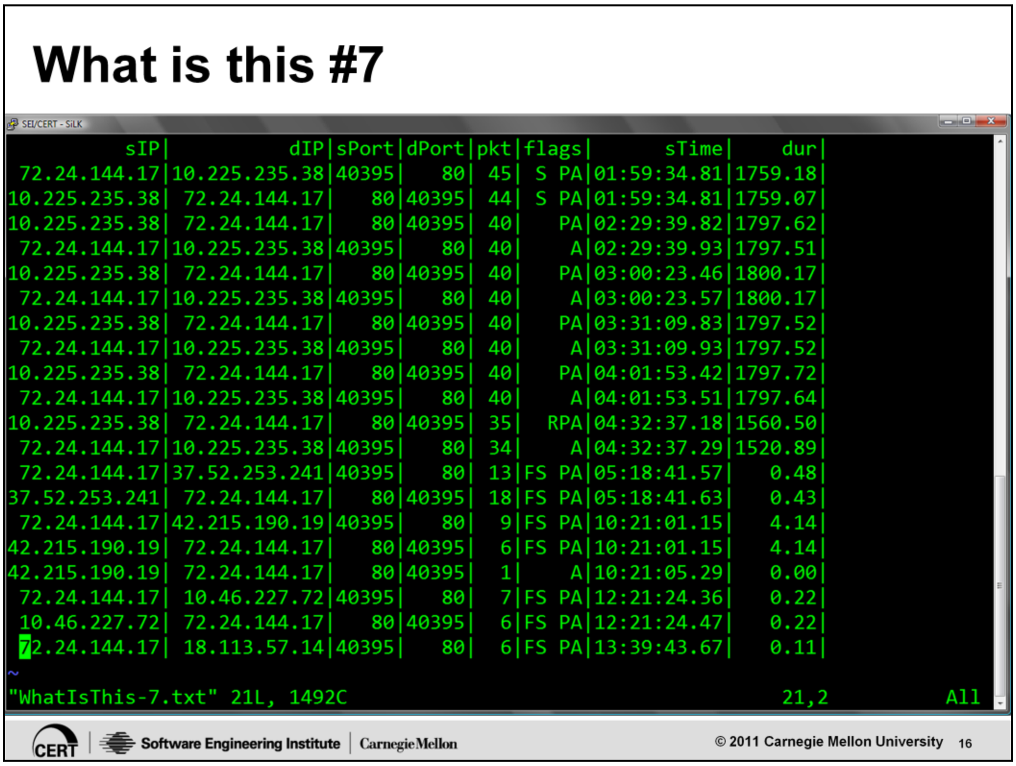
What do we have here?

The first thing you'll probably notice about this traffic is the service port, 22. This is the port for SSH so these all appear to be SSH client / server flows.

All the flags are PSH-ACK--no SYN, FIN or RST flags--so this definitely looks like one very long lived TCP session.

Now look closely at the times. Each flow is 1800 seconds long—30 minutes, which is a common setting for the active timeout. Then look at the start time for each flow—they're almost exactly 30 minutes apart. However, there are two sensors in play here: one rolls the flow over at :26 and :56 minutes after the hour; the other one is on the hour.

This is normal SSH traffic for a long-standing, high-volume traffic.



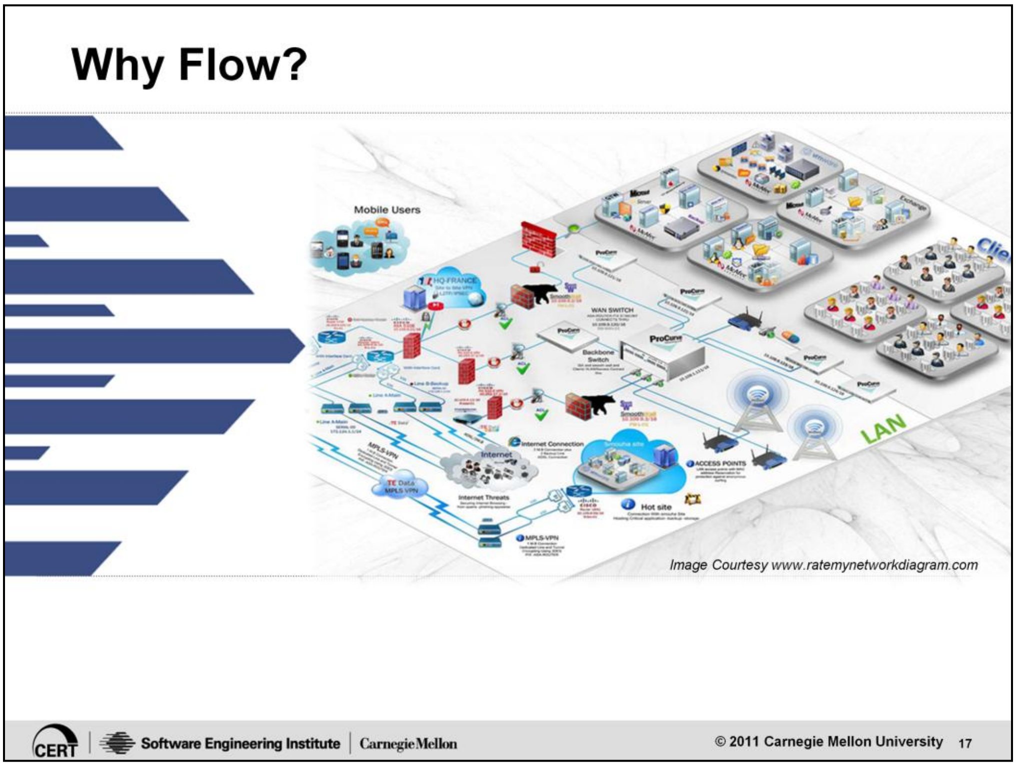
Here's the last example for you to take a look at. Don't worry that the IP addresses are in the 10.x network, this data has been anonymized.

This is port 80 web traffic. The first thing that should stick out immediately is that this is a very long TCP session for web traffic. Normally HTTP sessions deliver a web page and then close; occasionally streaming media will be delivered over HTTP, but that still rarely lasts for more than a few minutes. In this case, we see a single session that's been open from 2:00 until almost 5:00.

Now take a look at the rate packets are being sent: 40 packets every 30 minutes. That's slow, certainly not fast enough for streaming media.

Finally, look at the last set of connections. They're all new TCP sessions that occur over the course of a few hours, yet they all use the same ephemeral port. This looks very suspicious.

Overall, this does not appear to be normal HTTP traffic. What is it? We can't tell for sure, just that it does not fit the typical profile for web traffic.



That was interesting, but I can get all that from packet capture. Why bother with flow?

Sensor Types

Alerting

- Network IDS/IPS
- Host-based IDS

Data Collection

- Firewall, router
- Metadata collection
- Full packet capture

CERT | Software Engineering Institute | CarnegieMellon

Let’s start by considering the different types of sensors that might already be in your environment.

In general, alerting sensors feed some type of workflow so you can take action when something “interesting” happens.

On the other hand, data collection sensors just pull information off the wire and store it until you need it to investigate an event.

Flow fits in the “Data Collection” sensor; it’s similar to collecting firewall or router logs, metadata, or even full packet capture.

So if I have those data collection techniques already, why bother with flow?

Why Flow? Reason #1

Complaint: There’s no content!

Counter-complaint: That means privacy concerns are minimal. Many privacy-conscious organizations have legally approved flow collection.

The major complaint is that there’s no content with flow, so there’s no value.

That actually can be a significant strength if your collection efforts are hampered by privacy concerns. Even public service providers have been able to safely collect flow without violating privacy concerns.

Why Flow? Reason #2

Complaint: There's still no content!

Counter-complaint: Because there's no content, you can store ***lots*** of data. Using SiLK:

- Fully saturated 100mb link = 50Gb / month
- With a good storage back-end, you can easily query 50Gb of data in 10-20 minutes.

Details for SiLK storage and bandwith usage can be found in the SiLK Provisioning Worksheet at http://tools.netsa.cert.org/releases/SiLK-Provisioning-v204.xls

Another reason to use flow is to be able to store lots of historical data. Most flow compresses nicely and any traffic that doesn't (mostly DNS and scanning) can be easily trimmed from the long term data store if necessary.

Why Flow? Reason #3

Complaint: No, really, there's ***still*** no content!!

Counter-complaint: Content is a distraction for many problems.

- For many problems you need an index to get to the right content in your huge data store
- For other problems you need to search gobs and gobs of history
- Encrypted? So what!

That's the niche where flow lives

Believe it or not, there are a lot of problems you face that really don't need content. Or problems that you can't ask to your entire full content data set; problems where you need a pointer to focus your query. That's where flow fits in nicely: you can do a broad sweep of your data set with flow, and then do a fine-grained query against your full content data store.

Since flow doesn't use content, it also has lots of advantages when analyzing encrypted traffic. Remember that VPN tunnel from earlier that wasn't being used?

Got a Question? Flow Can Help

- What's on my network?
- What happened before the event?
- Where are policy violations occurring?
- What are the most popular web sites?
- How much volume would be reduced with a blacklist?
- Do my users browse to known infected web servers?
- Do I have a spammer on my network?
- When did my web server stop responding to queries?
- Who uses my public DNS server?

We've spent time looking at where flow records come from, what data they contain, and seen how network traffic appears in flow data. Hopefully you now can begin to consider many different ways that flow can be used as part of your overall security suite. It can act as a tool to support forensic and historical analysis, it can monitor your infrastructure, help you gain situational awareness about your network, and locate some classes of security events.

The Big Picture

Q. Why Flow?

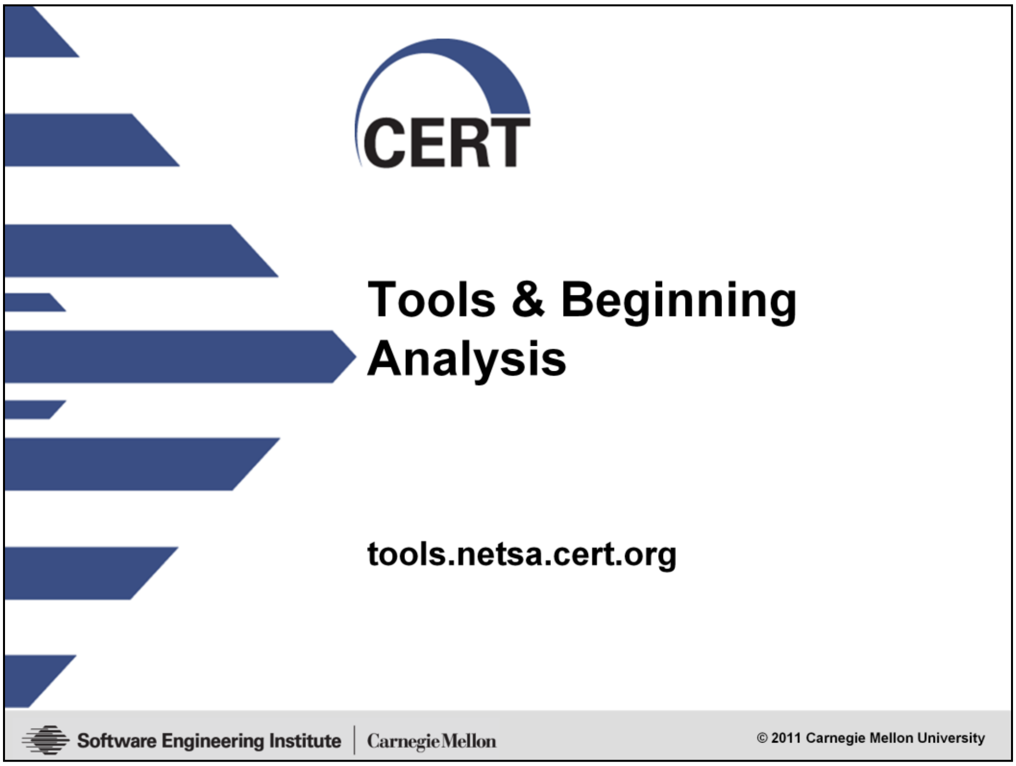
A. So you can get beyond signature-based detection

That's ***Network Situational Awareness:***

- Perceive
- Comprehend
- Predict

So at the end of the day, flow is your bridge between your data collection sensors and your alerting workflow. It's the place where you do things that are generally NOT signature-driven (although that's starting to change), the place where you really begin to understand what's on your network.

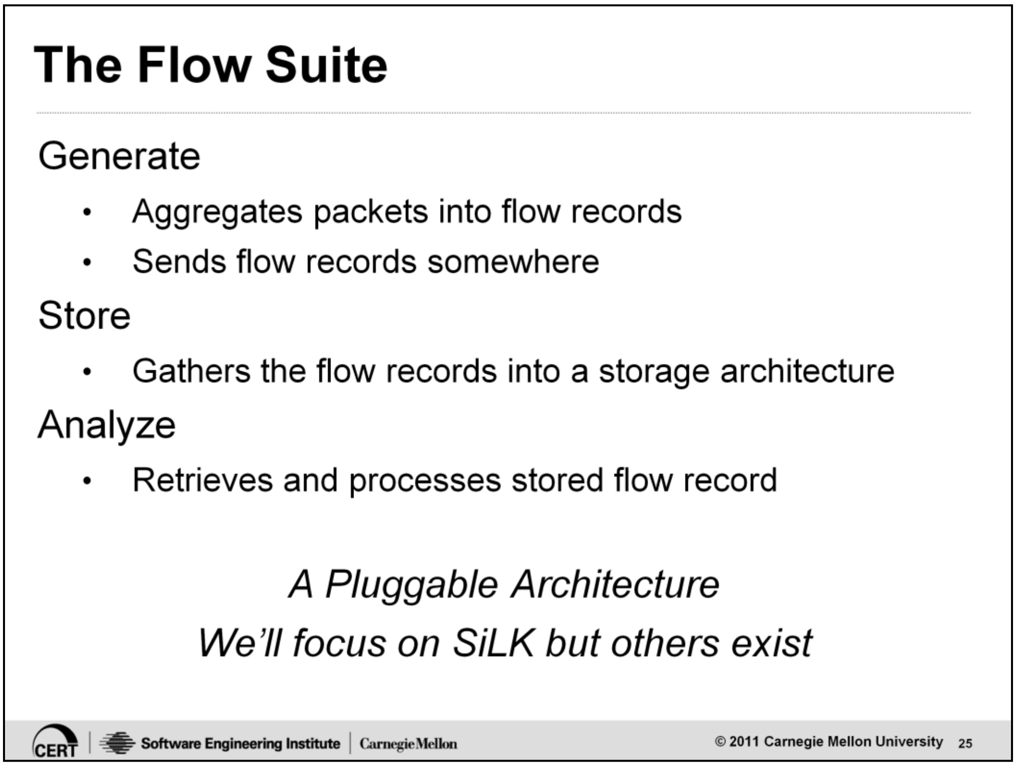
For more information about situational awareness, see Mica Endsley's paper "Toward a Theory of Situation Awareness in Dynamic Systems", Human Factors, 1995, 37(1), p32-64.



Argus is another popular open source flow collection and analysis package. Commercial products exist as well. There's even a module you can bind to a Linux Ethernet adapter to generate flow.

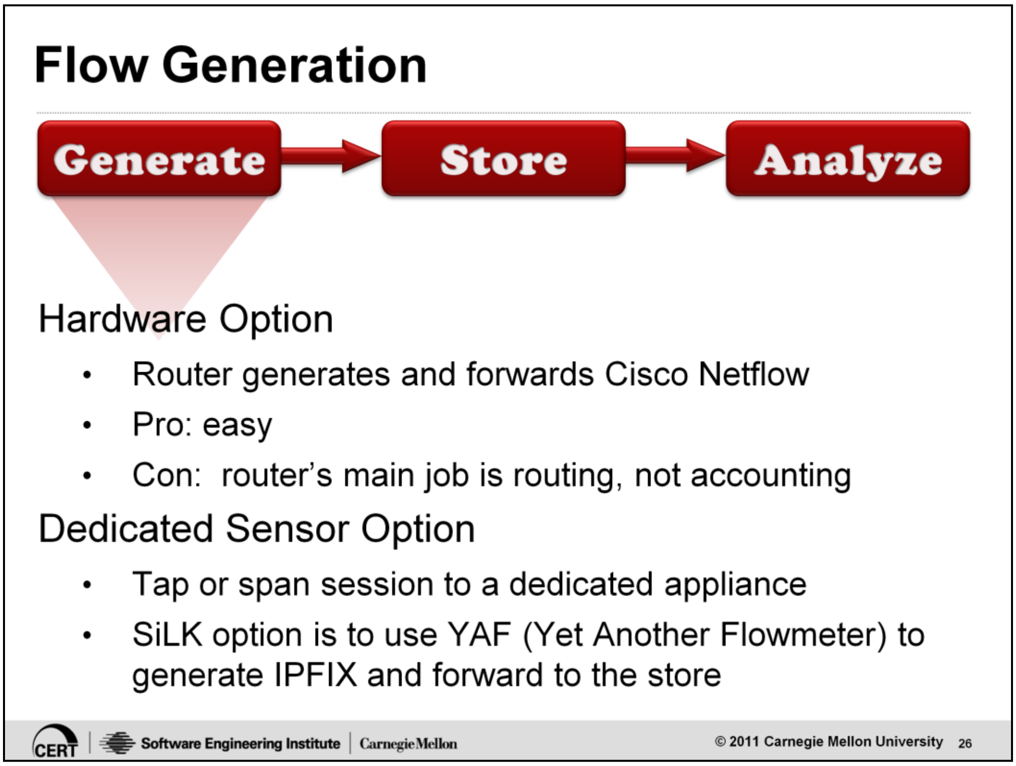
For the rest of this class we'll be looking specifically at the SiLK toolkit. It's very mature (over a decade of large installations), scalable (used by large organizations and service providers), actively maintained and it's open source.

Let's dig into some of the tools in the SiLK toolkit and some things you can do with them.



A complete flow implementation requires three things: something that generates flow records from packets on the wire, something that turns the wire-format flow (or netflow) records into a disk format and something that allows you to analyze the on-disk flow records.

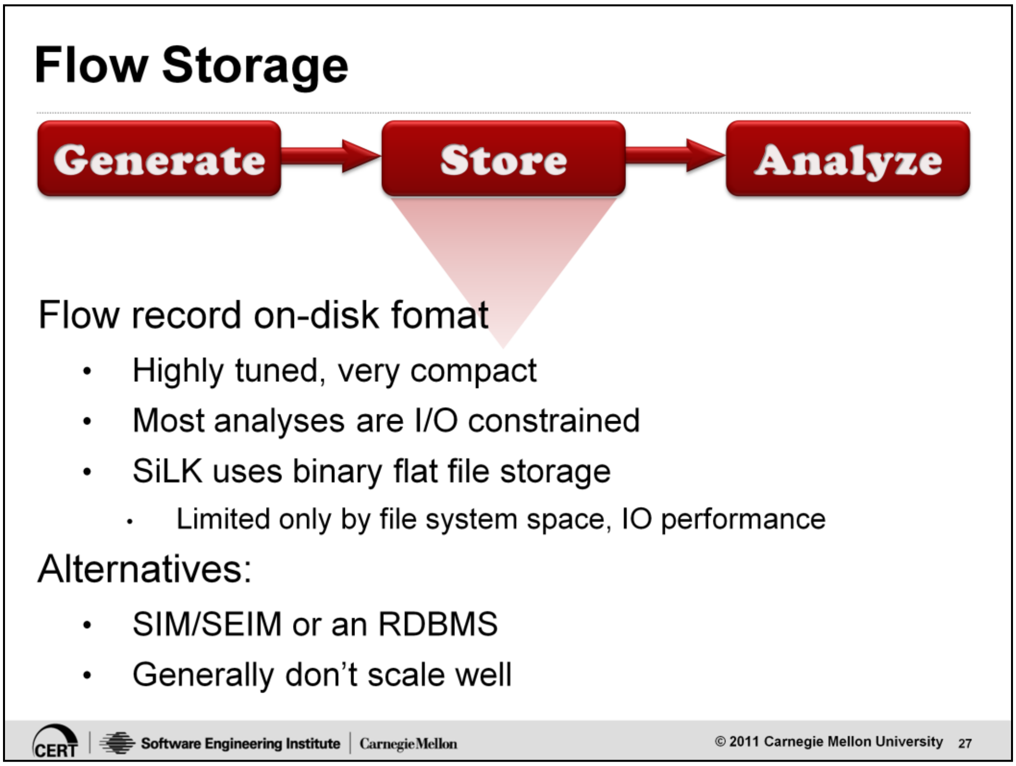
Commercial solutions based on netflow often will combine two or more of these things together, but it's important to keep them all in mind as you plan for interoperability between sensors and security event management systems.



Most commercial routers can generate flow, so it might be an easy matter to simply turn on flow accounting and have records generated for you. However, remember that a router’s primary job is to route packets, not to count flow. As the router becomes overburdened—particularly in a denial-of-service condition—the router will stop generating flow.

One alternative to reduce the load of generating flow records is to use sampling. For sampled flows, the router will only count every 100th or 1000th packet. This works fine for determining large-scale network trends, but it tends to mask many of the network behavior details that make flow valuable.

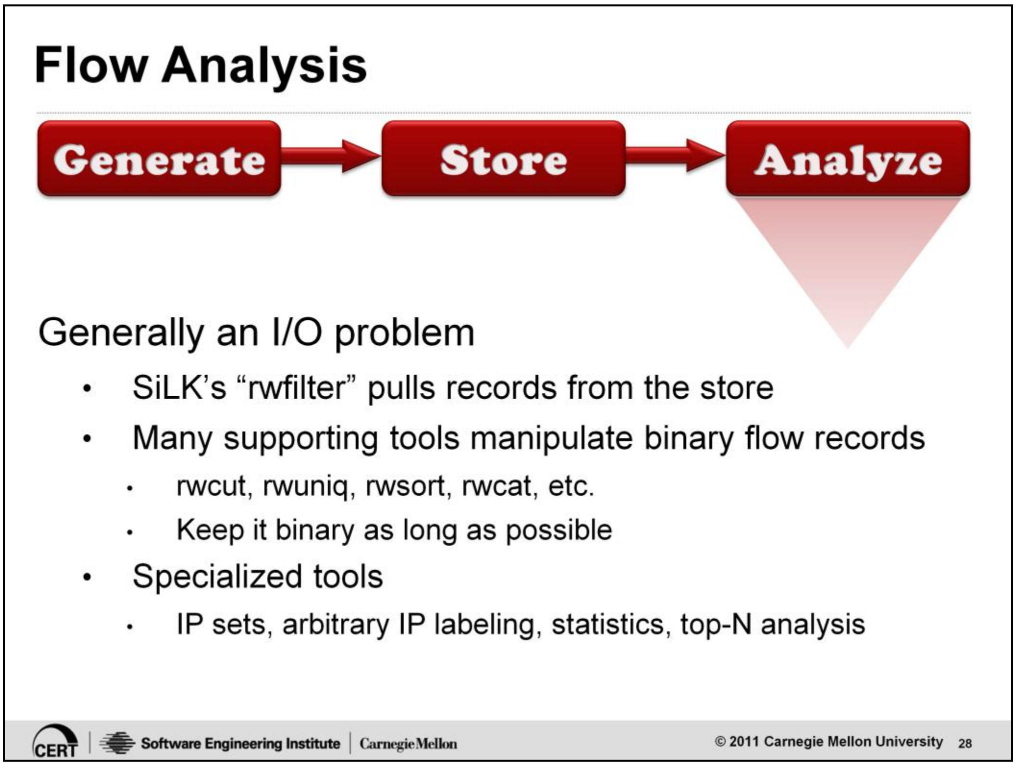
Another alternative is to use a dedicated sensor that exists specifically to monitor traffic and generate flow. The SiLK suite includes “Yet Another Flowmeter” (YAF), a service designed specifically to create flow from a network tap.



The flow records are sent to a collector in Cisco Netflow or IPFIX format, and the collector is responsible for storing the records on disk.

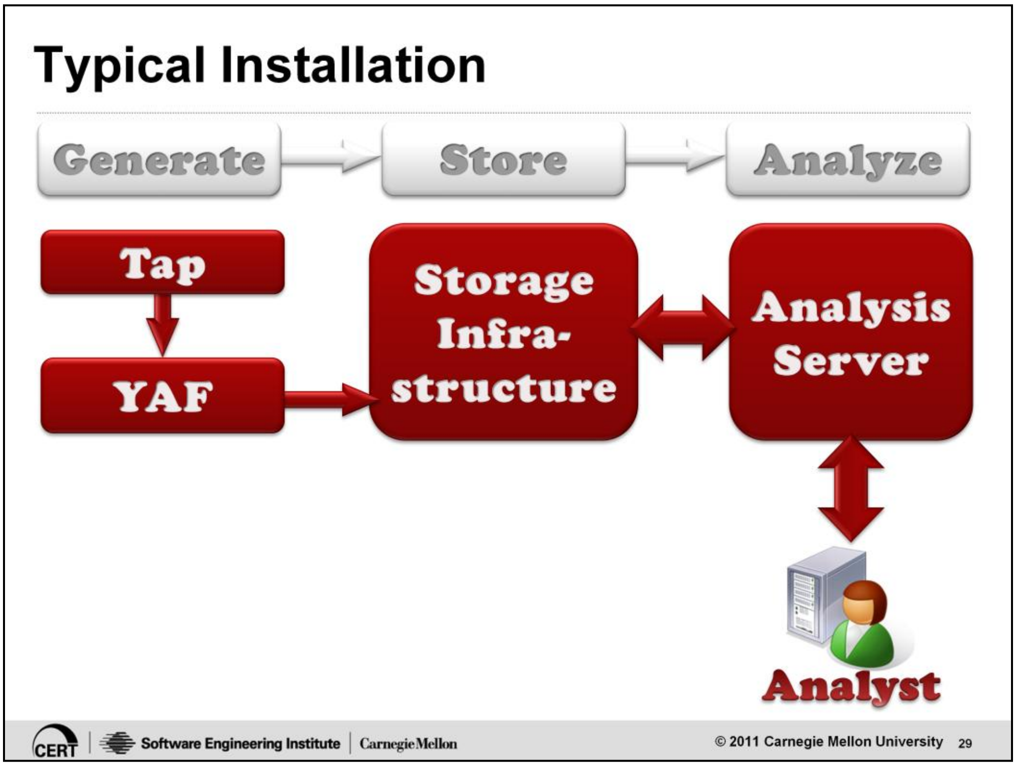
SiLK uses a very tightly compressed proprietary data structure for storing flow on disk. Since analysis tasks are often bound by the amount of time it takes to retrieve data from disk, it's important to keep the disk footprint small.

It's also possible to store flow records directly in your security information manager or in a relational database. That's a great way to correlate flow with other events, but it generally will not scale well in larger organizations.

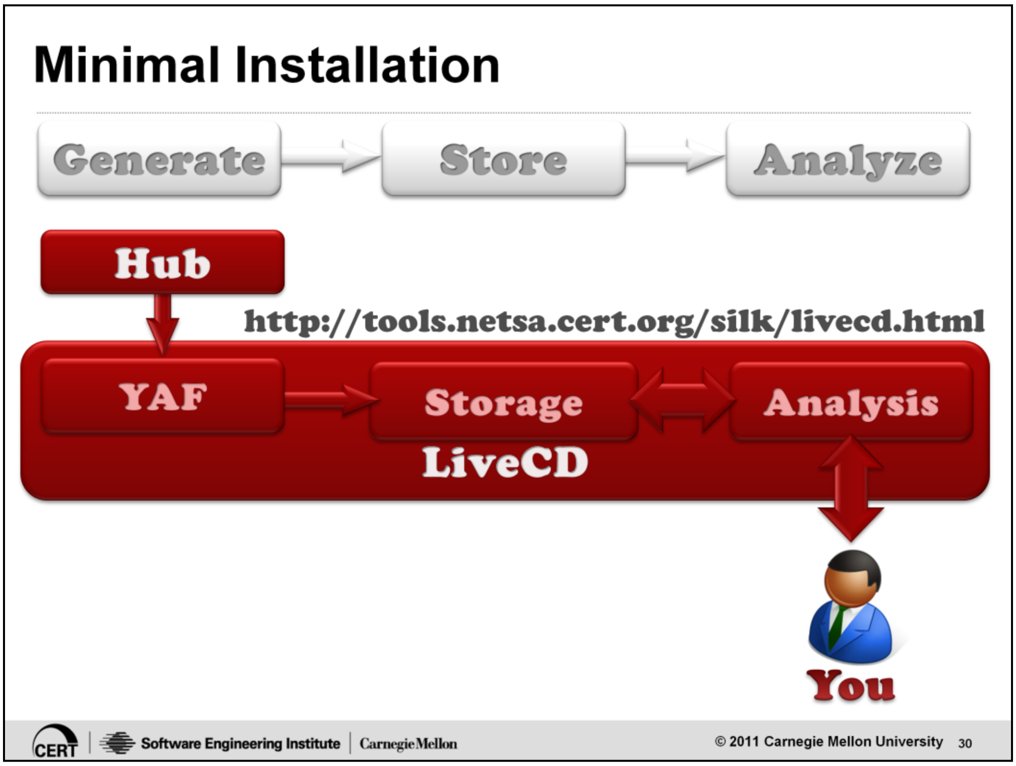


That leads us to analysis of the stored flow data. If you've used SiLK as your storage back end, you can take advantage of a number of specialized analysis tools in the suite. The tools are designed to keep flow data in the compressed binary form for as much of the analysis as possible—although you may be tempted to print out flow records as text and bring them into Excel, you'll be much more successful in large scale analysis by keeping the data in binary.

Although the analysis tools are mature, you should still think of them as middle-tier tools. They're generally all Unix command-line driven and produce text output.



The typical large organization will have three distinct tiers for its flow collection and analysis setup. A dedicated tap will feed a flow sensor, the sensor will send data to a distributed storage infrastructure, and the analysis server will access data in the storage cloud.



If you're a tinkerer, you can bypass all that complexity with the SiLK live CD. This CD has all the moving parts loaded into a single virtual image, just about all you have to do is to define a network interface to begin sensing.

Now let's get back into some of the analysis that can be done once the flow infrastructure is set up.

Example #1: Poison Ivy

sIP	sPort	dIP	dPort	pro	pkts	bytes	ini	sess	sTime+msec	dur+msec
120.16.86.207	1637	235.185.173.131	3460	6	3	144	S	S	12:08:50.487	1.373
235.185.173.131	3460	120.16.86.207	1637	6	1	40		R A	12:08:50.722	0.000
235.185.173.131	3460	120.16.86.207	1637	6	1	40		R A	12:08:51.316	0.000
235.185.173.131	3460	120.16.86.207	1637	6	1	40		R A	12:08:52.105	0.000
120.16.86.207	1638	235.185.173.131	3465	6	3	144	S	S	12:09:52.093	1.444
235.185.173.131	3465	120.16.86.207	1638	6	1	40		R A	12:09:52.313	0.000
235.185.173.131	3465	120.16.86.207	1638	6	1	40		R A	12:09:53.020	0.000
235.185.173.131	3465	120.16.86.207	1638	6	1	40		R A	12:09:53.771	0.000
120.16.86.207	1639	235.185.173.131	3666	6	3	144	S	S	12:10:53.774	1.670
235.185.173.131	3666	120.16.86.207	1639	6	1	40		R A	12:10:54.025	0.000
235.185.173.131	3666	120.16.86.207	1639	6	1	40		R A	12:10:54.787	0.000
235.185.173.131	3666	120.16.86.207	1639	6	1	40		R A	12:10:55.661	0.000
120.16.86.207	1642	235.185.173.131	3460	6	3	144	S	S	12:14:37.755	9.004
120.16.86.207	1643	235.185.173.131	3465	6	24	2040	S	F PA	12:15:58.791	513.393
235.185.173.131	3465	120.16.86.207	1643	6	40	16809	S A	PA	12:15:59.026	513.398
120.16.86.207	1649	235.185.173.131	3460	6	26	3688	S	PA	12:24:32.184	95.440
235.185.173.131	3460	120.16.86.207	1649	6	38	23113	S A	PA	12:24:32.424	95.550
235.185.173.131	3460	120.16.86.207	1649	6	94	6526	PA	PA	12:28:51.561	1789.016
120.16.86.207	1649	235.185.173.131	3460	6	55	6664	PA	PA	12:28:51.674	1788.529
235.185.173.131	3460	120.16.86.207	1649	6	81	5207	PA	PA	12:59:25.227	1800.622
120.16.86.207	1649	235.185.173.131	3460	6	41	3608	PA	PA	12:59:25.265	1800.619

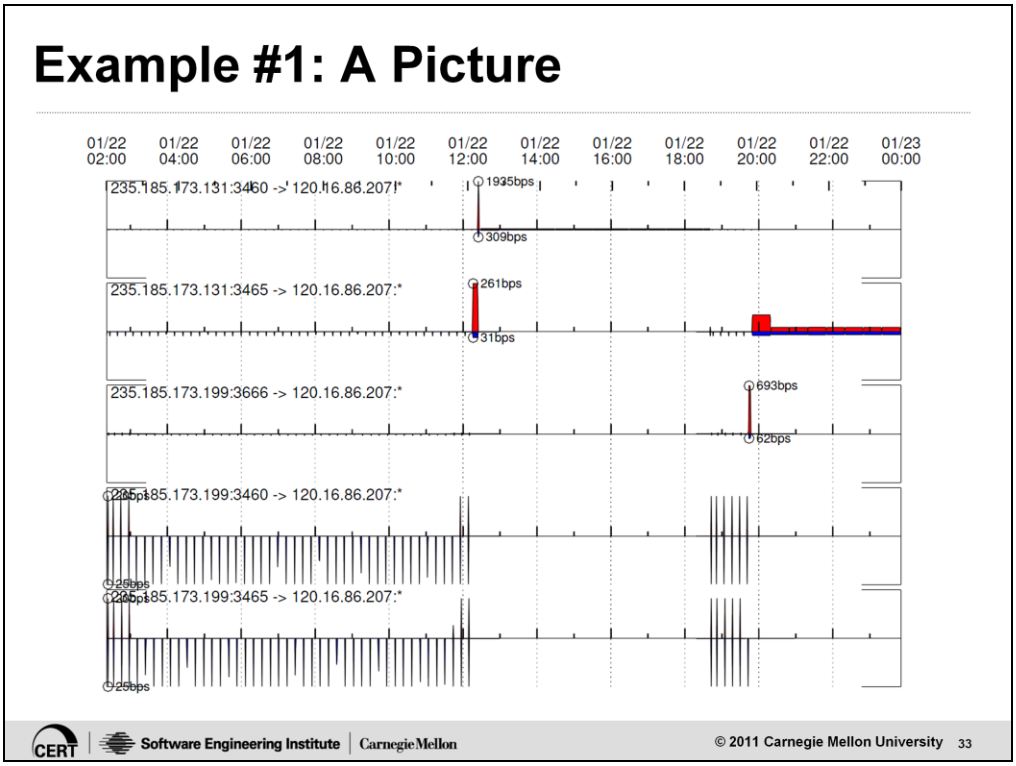
...

This showed up one day while watching the default poison Ivy port (3460). Can you explain what's going on here based on what you know about poison ivy? What characteristics might help you decide if this is botnet command/control traffic or just routine traffic on a non-standard port?

120.16.86.207 might have an implant that's trying to connect to 235.185.173.131, but it looks like we're missing the first SYN packet in this screen shot. It seems to be retrying the connection attempt every 60 seconds or so.

There's a point where the connection succeeds, there's a lot of data sent from the server (182) back to the client (117), and then the connection stays open for over an hour.

Even with those features, it's difficult to get a feel for what's going on here. Let's take a look at how the activity proceeded over a longer time period.



Here’s an example of essentially the same traffic plotted over a 24-hour time span. Now can you spot the beacon? And what do you think happened at 12:15? How about 18:30 and 19:45?

Originally we thought there was a beacon every 60 seconds; what’s actually happening are three retries 60-seconds apart, and then a rest for 30 minutes.

This plot was generated using the “stripplot.py” python script with the command

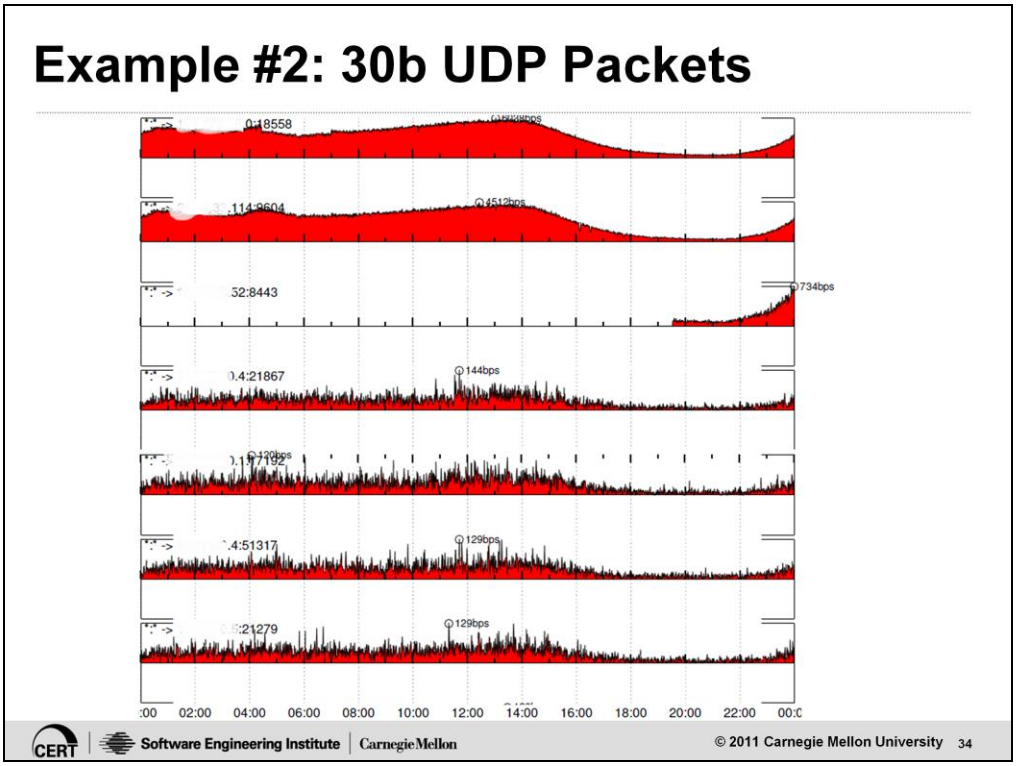
```
./stripplot -v -count=6 -fields=* controller.rw
```

This script uses SiLK, gnuplot and ghostscript to generate simple time-series flow visualizations as a .pdf file. It’s available here:

https://tools.netsa.cert.org/confluence/display/tt/Strip+Plots

And there’s a presentation on how to use it here:

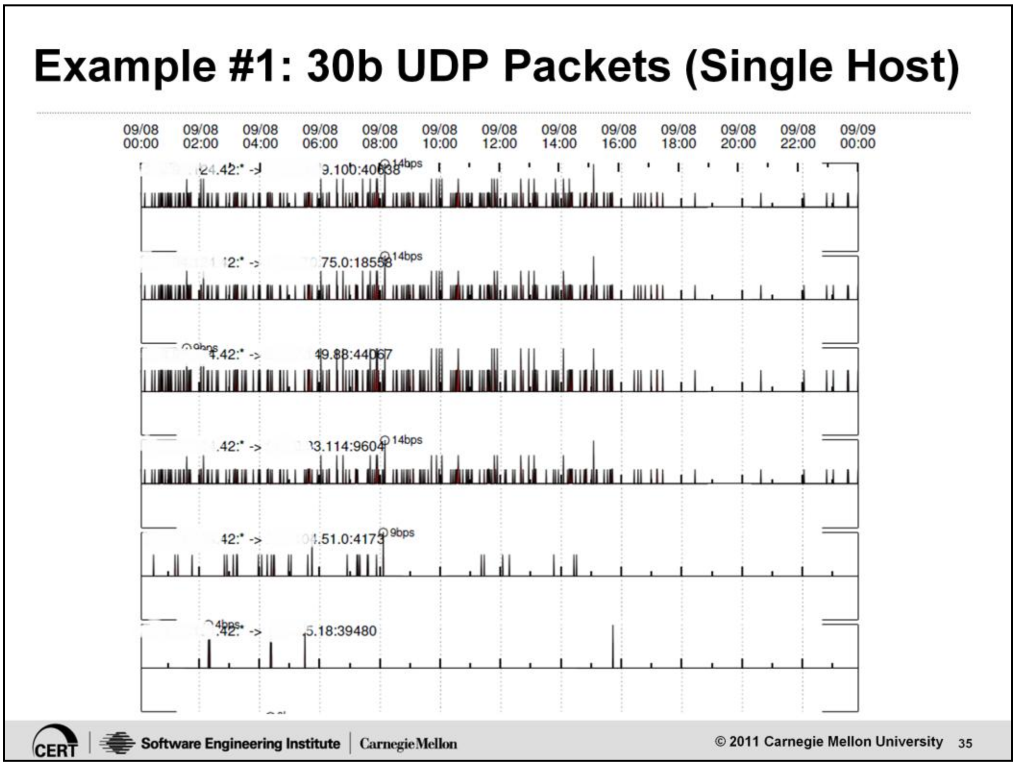
https://tools.netsa.cert.org/confluence/download/attachments/10027010/Faber_StripPlots.pdf?version=1&modificationDate=1263239683000



One day I started looking at 30 byte UDP packets transiting my network. I found tons of them coming from all over the place, destined for a small number of IP addresses in my network. Many were heading to addresses that ended in .0 or .4, but never for a live host. They were coming from hundreds of thousands of sources. Roughly 90% came from China, maybe another 8% from the Pacific Rim, and the remaining mostly from the United States.

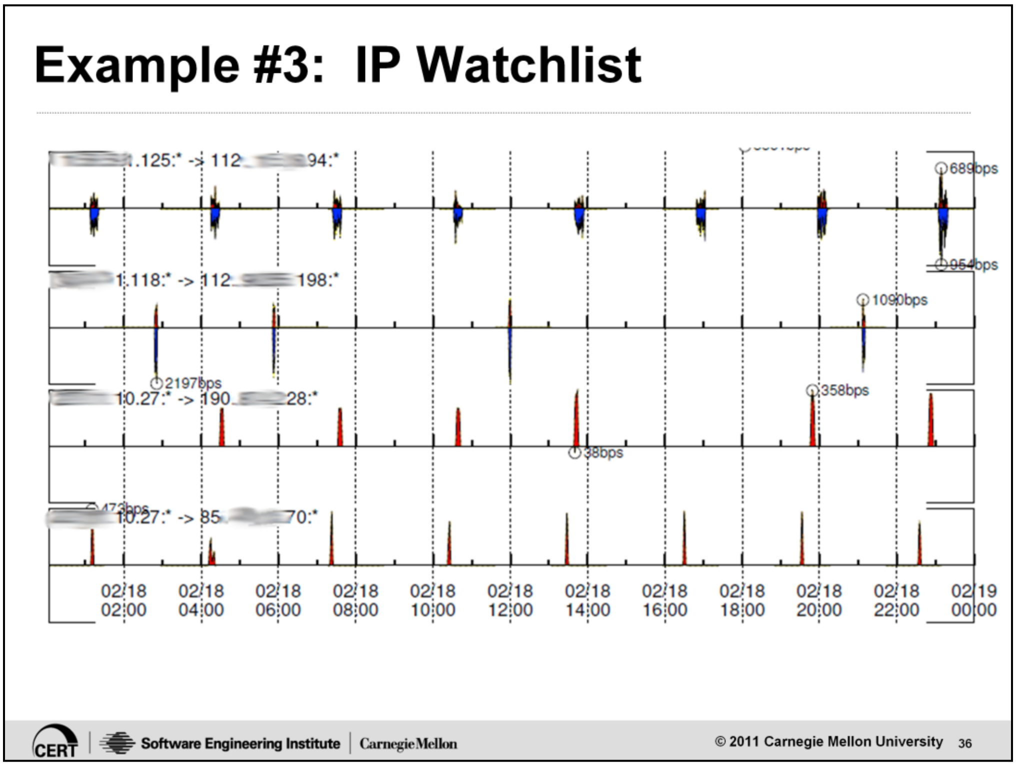
Look at the third strip: that's what I found really confusing. All at once tens of thousands of clients almost simultaneously started generating the same type of traffic.

Any idea what this might be?



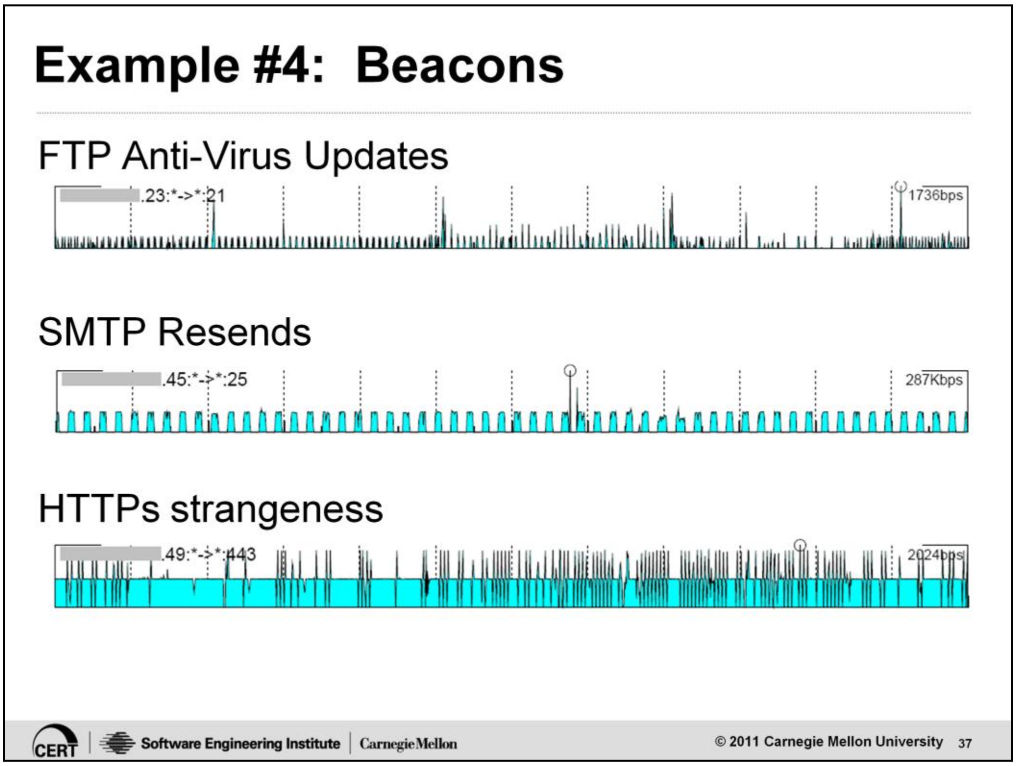
Here’s a view of the packets generated by a single IP address. Note how the source will hit up four destinations at a time. Whenver we see traffic like this we’re tempted to think the worst—that it’s some type of attack—but it just doesn’t have the hallmarks of a distributed denial-of-service attack or anything else particularly malicious.

So what is this? I’m still not sure, but I believe it’s some type of broken peer-to-peer software; likely peer-to-peer IP Television which is widely used in China. But I never found out for sure.



Can you spot the beacon? Any idea what this is?

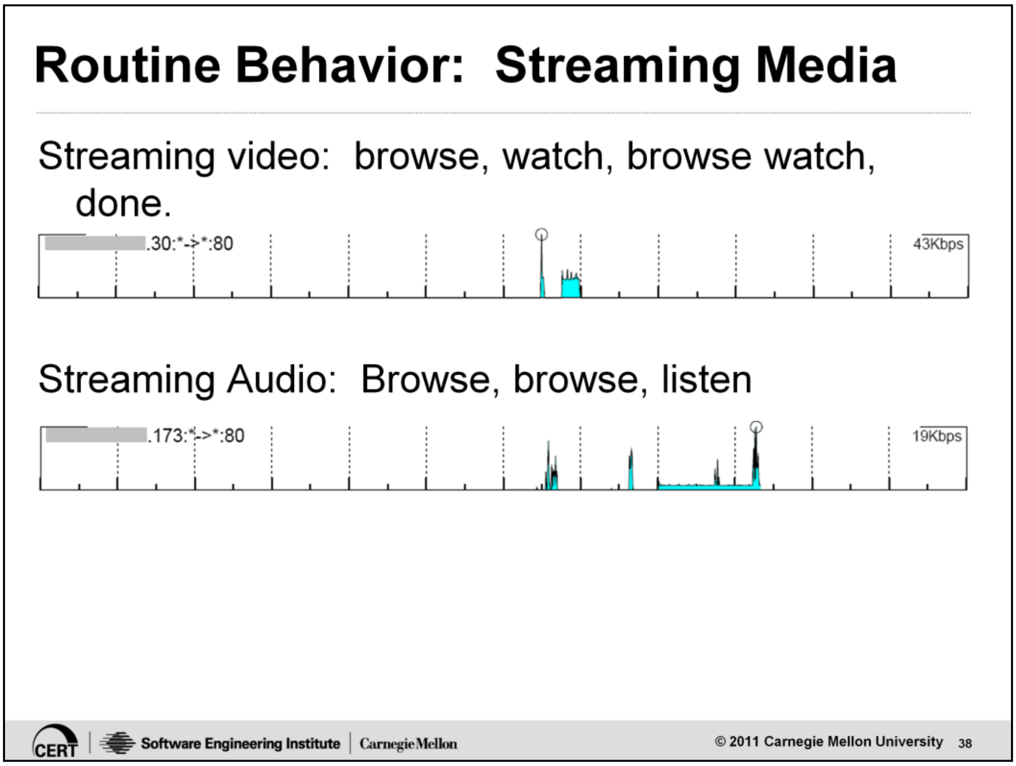
This is actually a view of traffic destined to known Conficker sinkholes. Review what you know about conficker—can you explain what's going on in these plots?



All beacons are not malicious. Here are some examples of two benign beacons. The first one is for anti-virus updates; in this case it looks like the update check is about every 10 minutes. This might represent updates for multiple clients behind a single NAT'd IP address. Can you tell when the clients are shut off for the evening?

The second strip shows some aggressive SMTP retries. There are a number of scenarios where email can get caught in a resend loop for a few days, can you think of any?

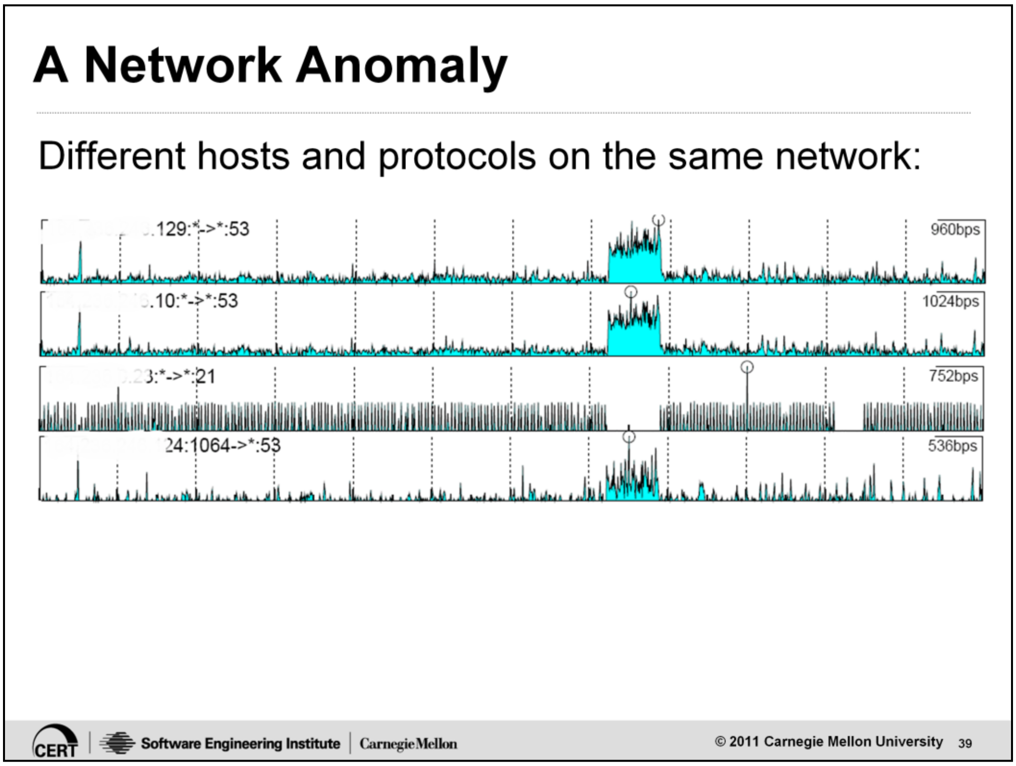
The final plot is just plain odd HTTPS traffic. It's relatively slow and flat, it looks kind of like a stale HTTPS channel...but it shouldn't stay active for a whole day!



Here’s a comparison of traffic for streaming media, typically the largest consumer of bandwidth on your network.

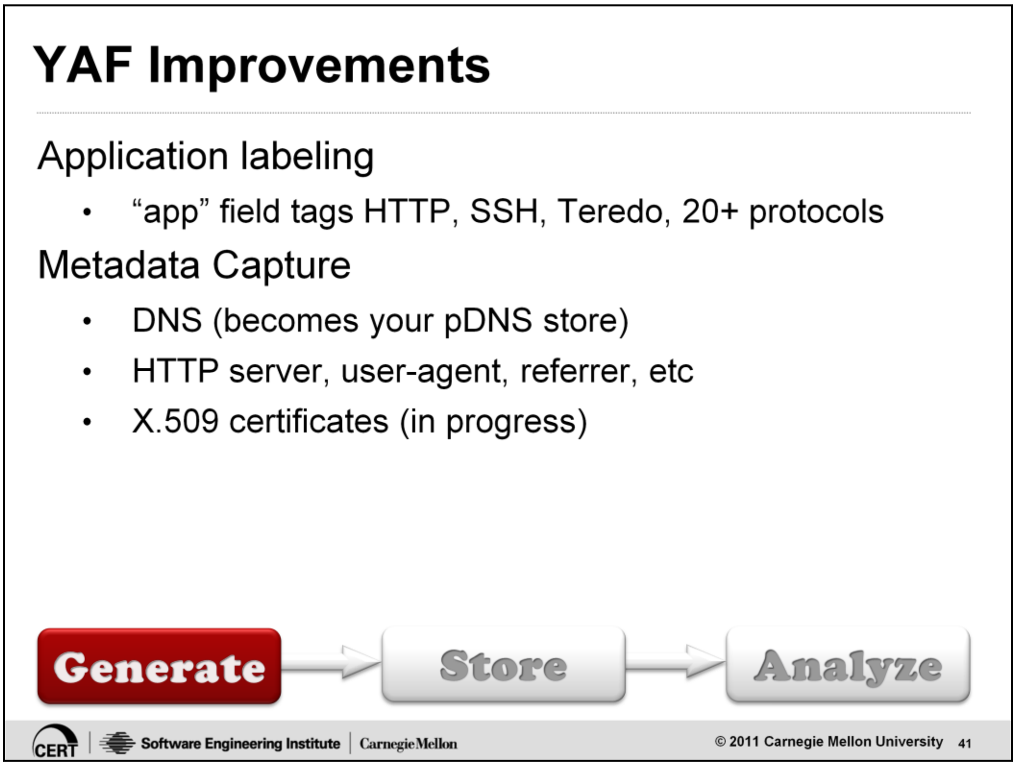
Note how streaming video—youtube in this case—shows a browse, watch, browse, watch pattern and the videos are all rather short.

Contrast that with the streaming audio example in the second strip. Bandwidth is smaller, but you see a steady utilization that covers a few hours.



And what’s going on here? I’m not quite sure, but there’s definitely an anomaly. Probably there was a network outage that resulted in an increased volume of DNS retry attempts.

Let’s take a brief look at some of the things that are on-the-way for the SiLK tool suite.

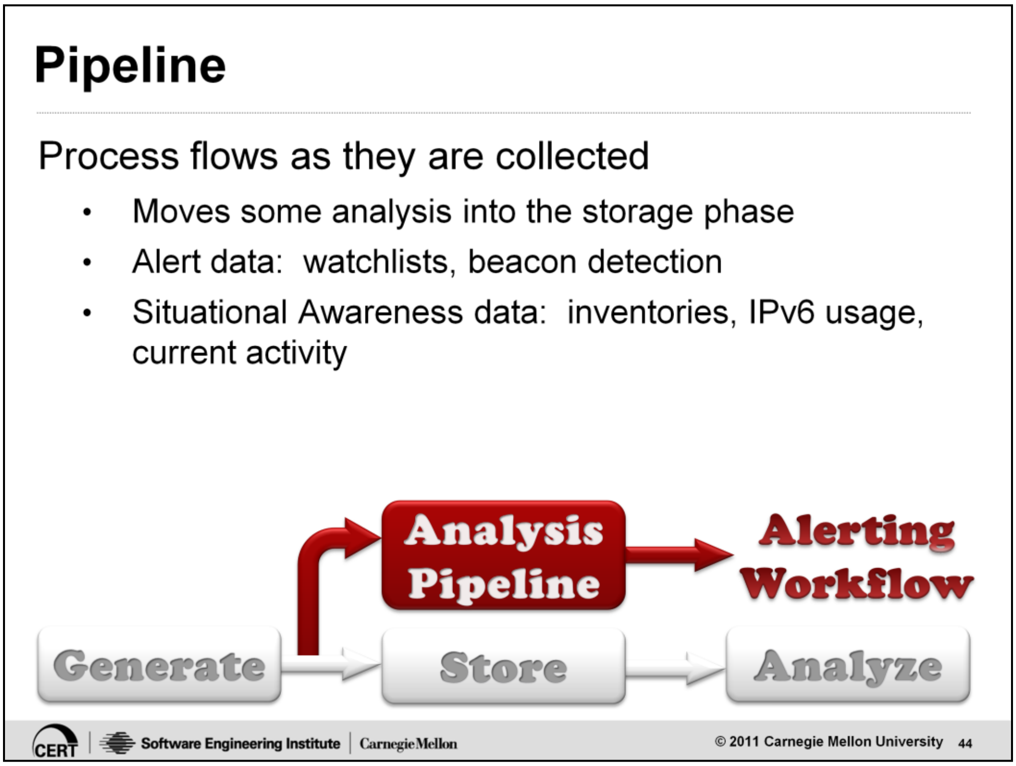


Recall that YAF is the tool that counts packets and creates flow records. YAF includes an "application labeling" feature that allows you to assign an integer to a flow based on deep packet inspection. This is very useful to identify protocols—for example, a relatively simple regular expression can identify HTTP traffic running on any port and assign it an application label of "80".

Since YAF is inspecting packets, it has been re-engineered to pull deeper metadata out of the conversation. Since IPFIX allows for more generic passing of information about a flow, YAF can pull some metadata off the wire and add that to the flow record. Eventually YAF will become a passive DNS collector, HTTP monitor and pull metadata out of any number of protocols.

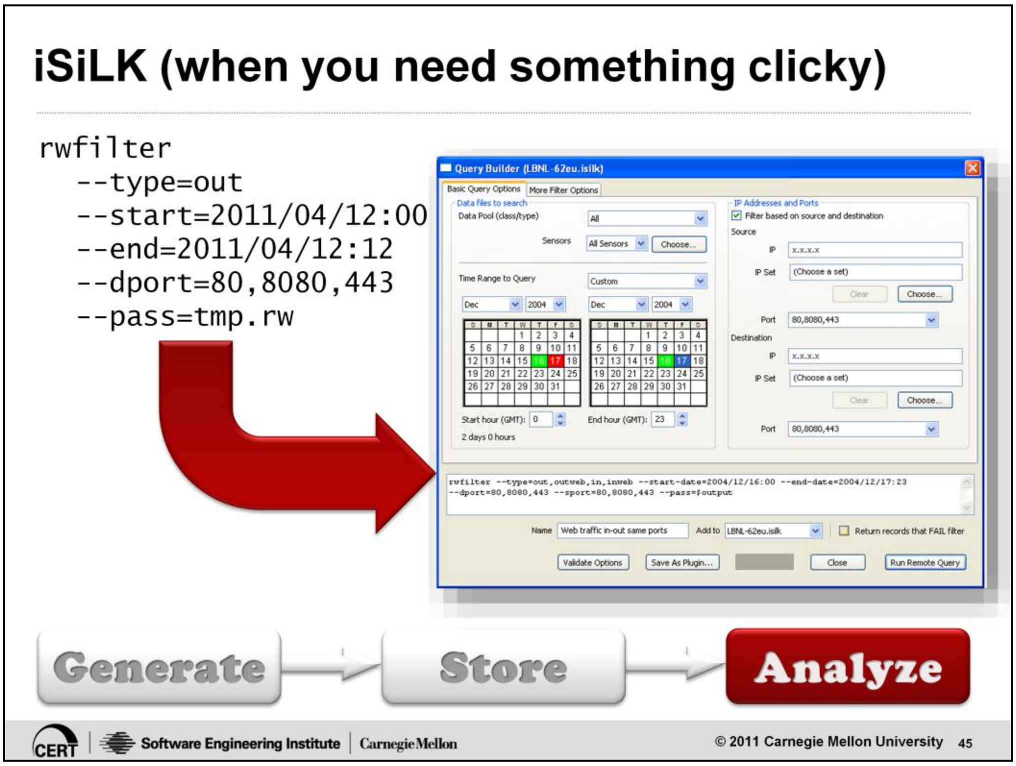
Although IPFIX allows collection of metadata, a mature storage solution for the data does not yet exist. yInspector seeks to fill that gap with flexible storage and query of IPFIX metadata.

[intentionally blank]

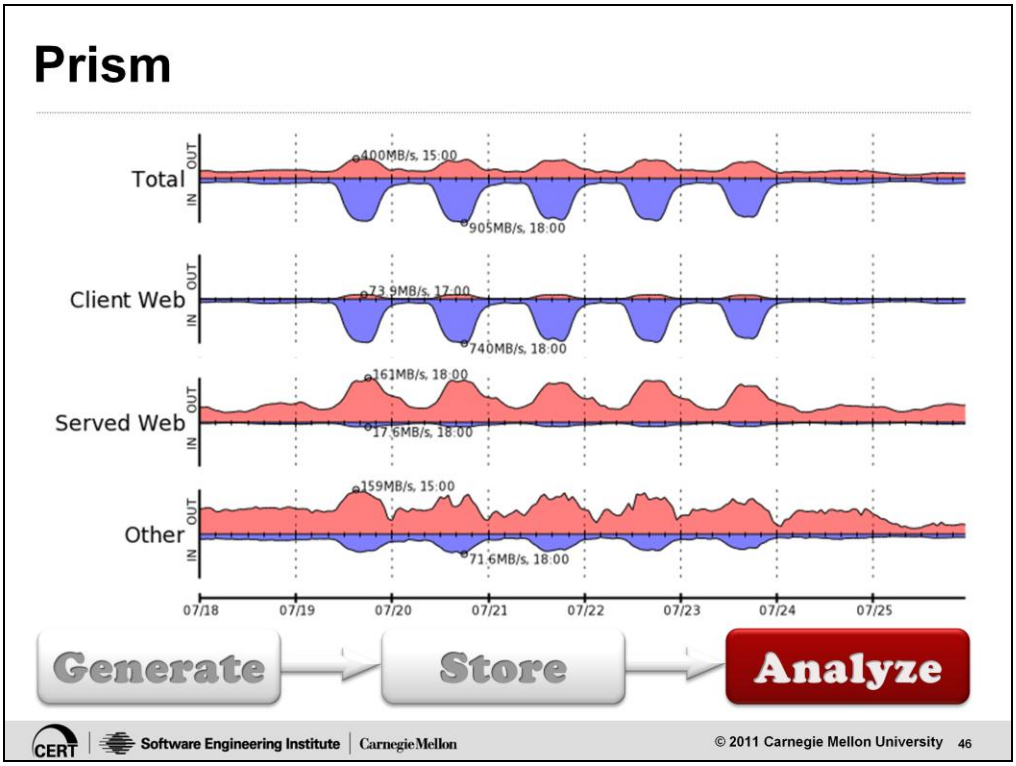


Pipeline is a new and significant improvement that will allow flow analysis to feed directly into the alerting flow in real time—but it won't require you to throw all your flow records into your security correlation engine.

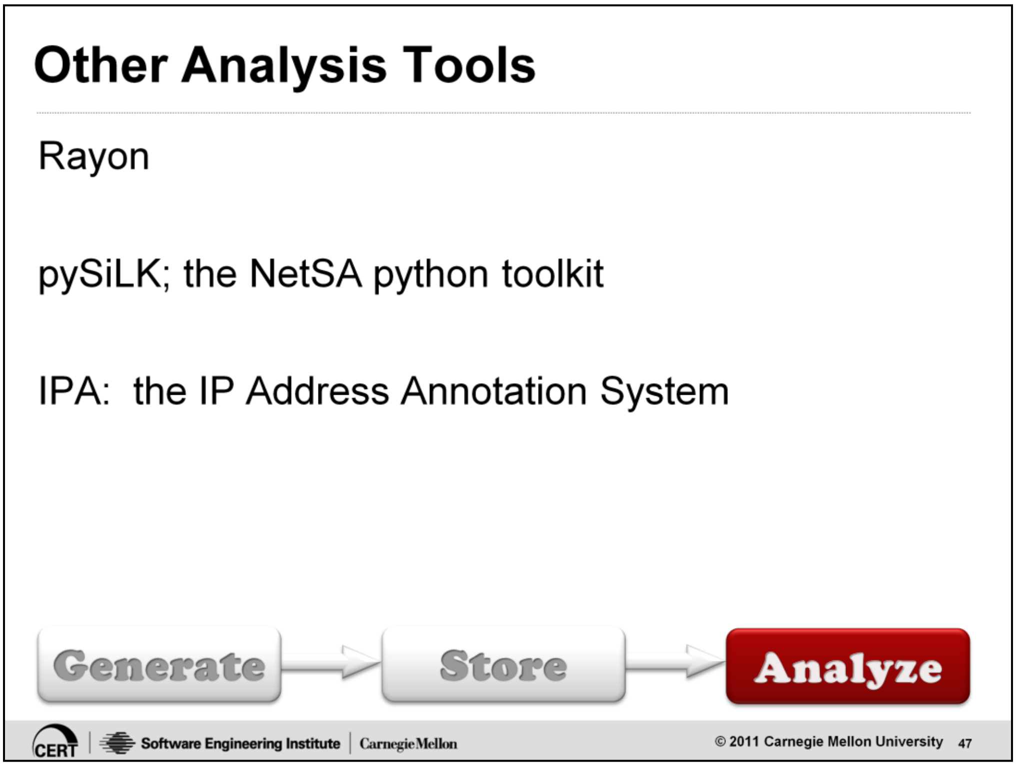
Analysis pipeline takes a streaming feed of events, maintains state, performs analytics and feeds an alerting library.



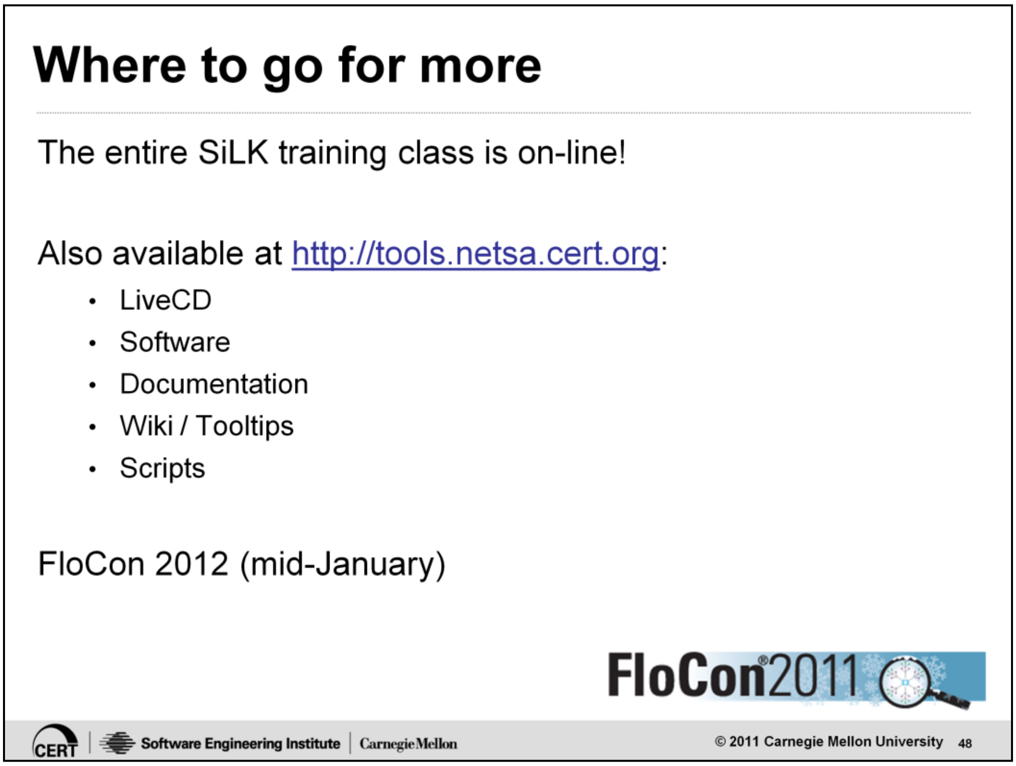
As mentioned at the beginning, all of the SiLK analysis tools run from the Unix command-line. iSiLK is a graphical user interface that will generate all the commands for you, display the output in a spreadsheet and produce simple visualizations.



Prism is a situational awareness tool used to carve traffic up into flexible bins and display long term trends.



Stay tuned, there’s much more in the works!



For the SiLK training class go to https://tools.netsa.cert.org/ and select the “Online Training” link in the left column. This brings up the public Virtual Training Environment course on “Using SiLK for Network Traffic Analysis”. This five-hour course covers most of the SiLK analysis tools and techniques and includes a hands-on lab.

You can find the SiLK LiveCD at https://tools.netsa.cert.org/silk/livecd.html or by clicking on “SiLK” under the Projects section and selecting LiveCD from the menu.

[intentionally blank]