CSEN1001

Computer and Network Security

Mervat AbuElkheir
Ahmad Helmy
Mohamed Abdelrazik

Lecture (11)

Attacks and Countermeasures

Triton is the world's most murderous malware, and it's spreading

Stu Sjouwerman

In the summer of 2017, a petrochemical plant in Saudi Arabia experienced a worrisome security incident that cybersecurity experts consider to be the first-ever cyber attack carried out with "a blatant, flat-out intent to hurt people." The attack involved a highly sophisticated new malware strain called Triton, which was capable of remotely disabling safety systems inside the plant with potentially catastrophic consequences.

Luckily, a flaw in the Triton code triggered a safety system that responded by shutting down the plant. If it hadn't been for that flaw, the hackers could have released toxic hydrogen sulfide gas or caused explosions. As a result, employees of the plant and residents of the surrounding area could have been killed or injured.

Triton is almost certainly the work of state-backed hackers. While Iran was the initial suspect, later reports indicate that Russia may have been behind the attack, using spear phishing attacks to take over the network.

Since Triton was first discovered, cybersecurity firms have uncovered more attacks involving malware with similar traits, designed to take over safety systems. Triton has not been spotted in other potentially destructive attacks, but cybersecurity experts believe it is only a matter of time before the murderous malware will rear its ugly head again.

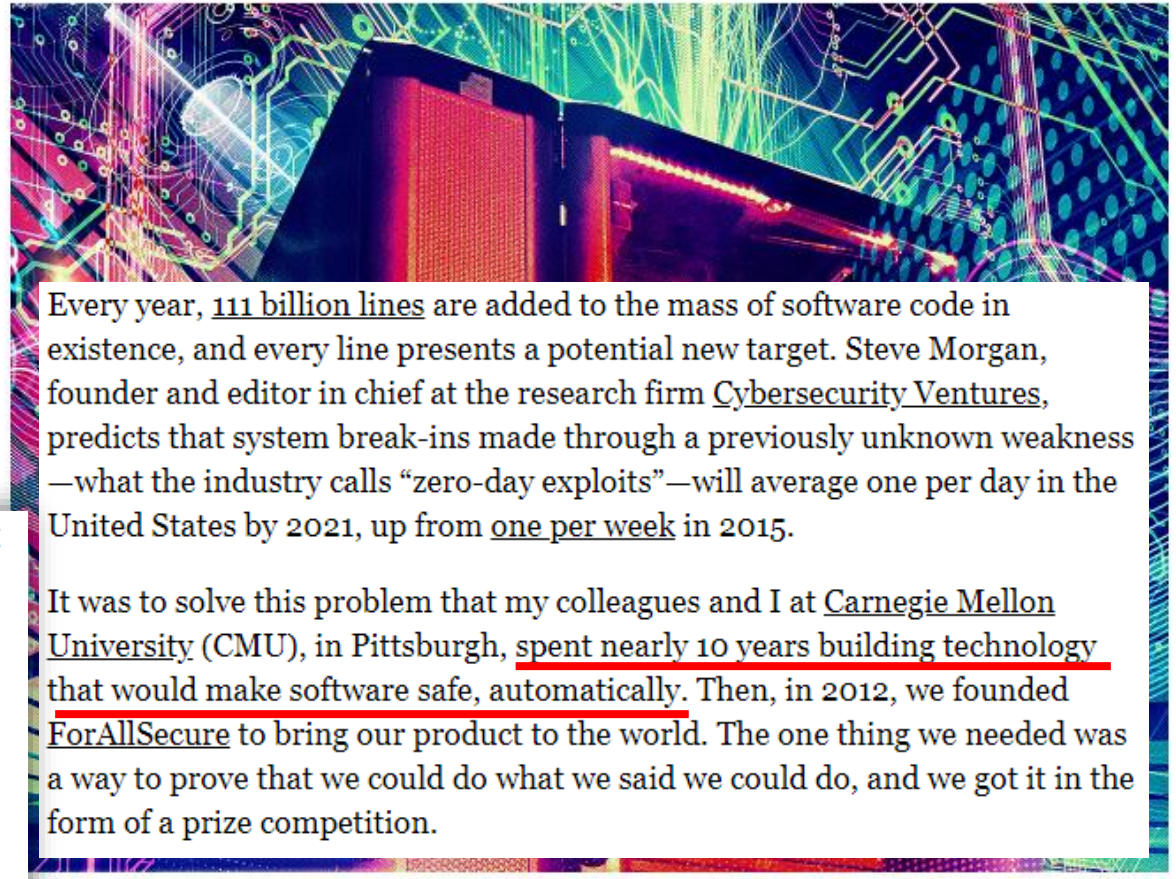
Full Artic
infrastru

Mayhem, the Machine That Finds Software Vulnerabilities, Then Patches Them

The machine triumphed in DARPA's Cyber Grand Challenge, where teams automated white-hat hacking

https://spectrum.ieee.org/computing/software/mayhem-the-machine-that-finds-software-vulnerabilities-then-patches-them

By **David Brumley**



Every year, 111 billion lines are added to the mass of software code in existence, and every line presents a potential new target. Steve Morgan, founder and editor in chief at the research firm Cybersecurity Ventures, predicts that system break-ins made through a previously unknown weakness—what the industry calls "zero-day exploits"—will average one per day in the United States by 2021, up from one per week in 2015.

It was to solve this problem that my colleagues and I at Carnegie Mellon University (CMU), in Pittsburgh, spent nearly 10 years building technology that would make software safe, automatically. Then, in 2012, we founded ForAllSecure to bring our product to the world. The one thing we needed was a way to prove that we could do what we said we could do, and we got it in the form of a prize competition.

oto-illustration: Sean McCabe

In 2014, DARPA announced the Cyber Grand Challenge as a two-year project with the goal of testing whether it was possible to develop AI systems that could find, verify, and patch software weaknesses. In 2015, some 100 teams entered the prequalification stage. In 2016, the top seven advanced to the grand championship finale, where they'd need to enter a full cyber-reasoning system—one that would not merely notice a problem but could also infer its nature. The champion would win US $2 million, and the second- and third-place finishers would get $1 million and $750,000, respectively.

Access Control

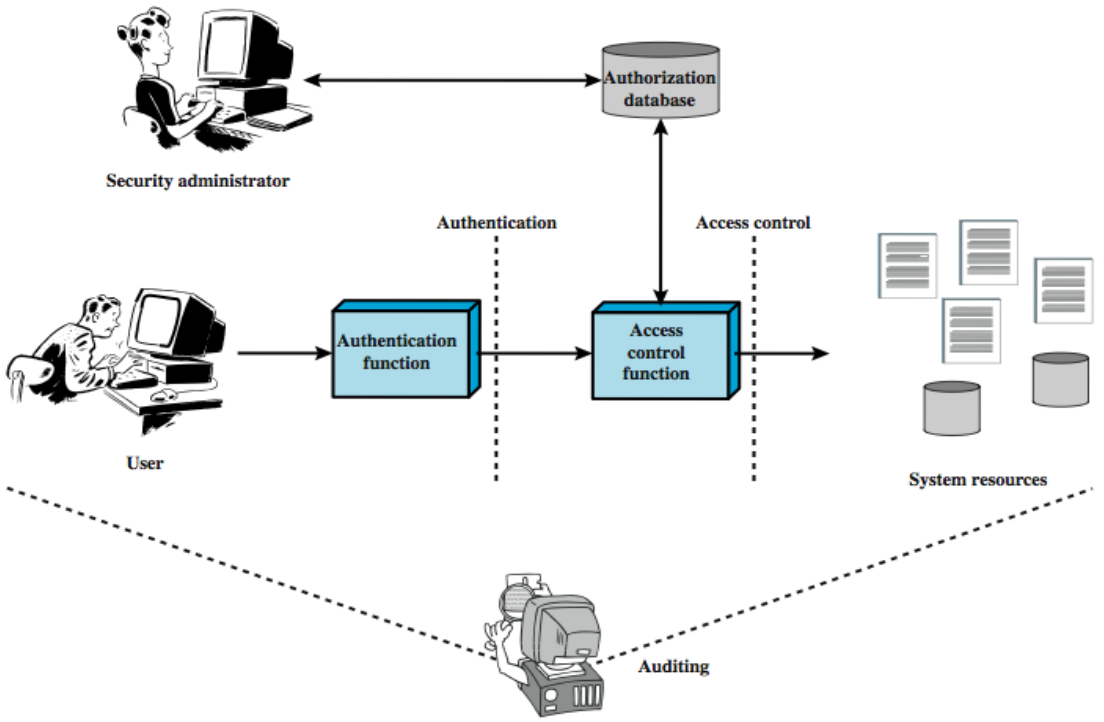
Access Control

“The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner“

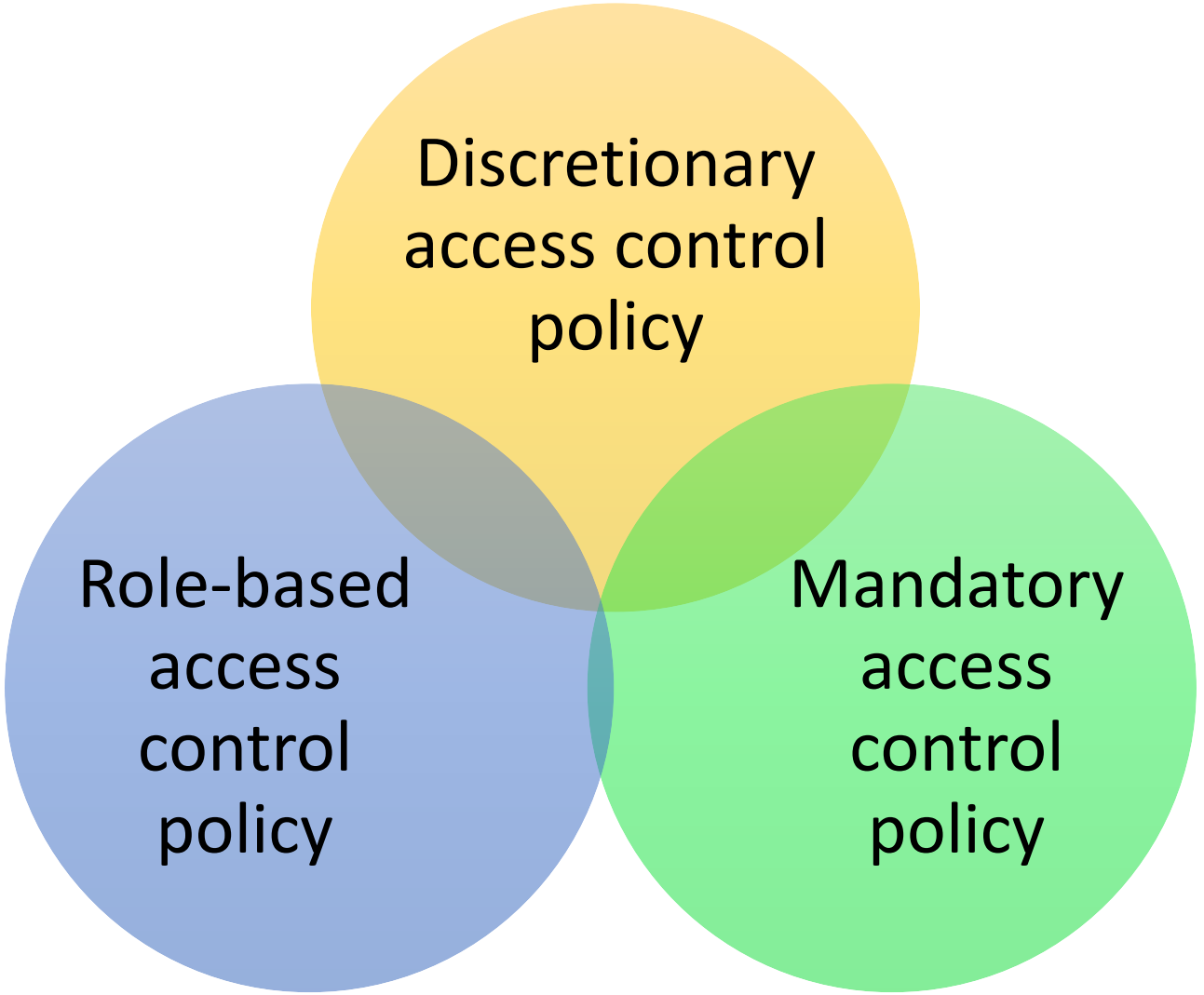
- Central element of computer security
- Assume we have users and groups
  - authenticate to system
  - assigned **access rights** to certain resources on system

Access Control Elements

- Subject - entity that can access objects
  - a process representing user/application
  - often 3 classes: owner, group, world
- Object - access controlled resource
  - e.g. files, directories, records, programs
  - number/type depend on environment
- Access right - way in which subject accesses an object
  - e.g. read, write, execute, delete, create, search



Access Control Policies



Discretionary Access Control

- Often provided using an access matrix
  - lists subjects in one dimension (rows)
  - lists objects in the other dimension (columns)
  - each entry specifies access rights of the specified subject to that object
- Access matrix is often sparse
- Can decompose by either row or column

Access Control Model

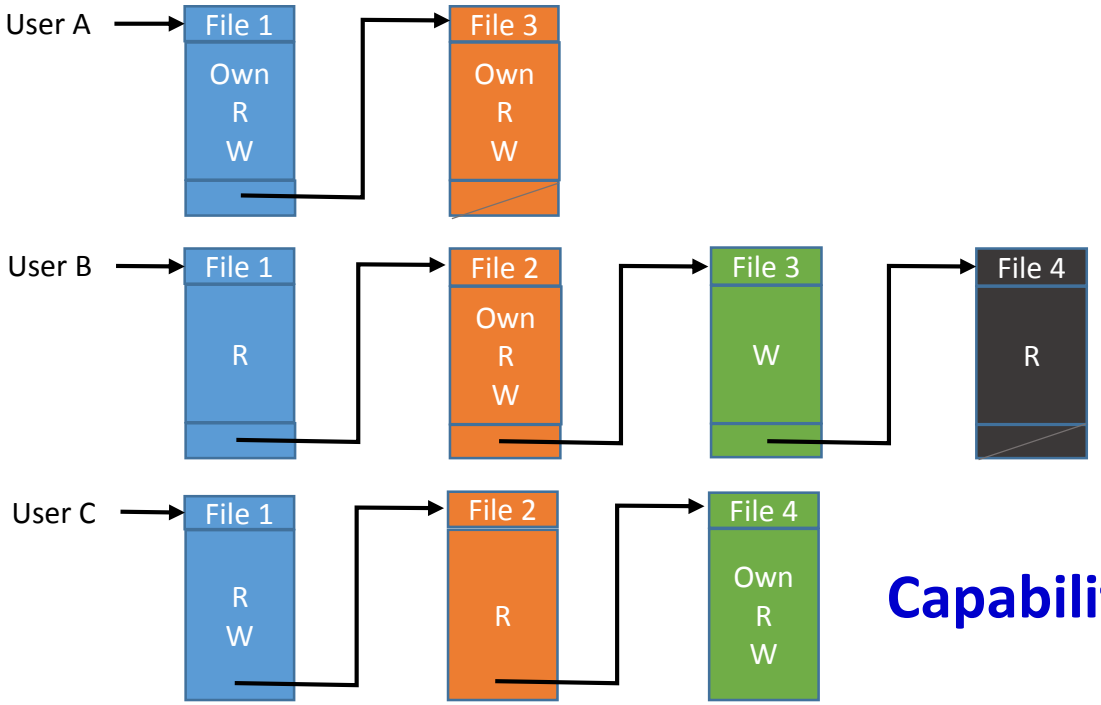
Extend the universe of objects to include processes, devices, memory locations, and other subjects

		OBJECTS								
		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_1	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

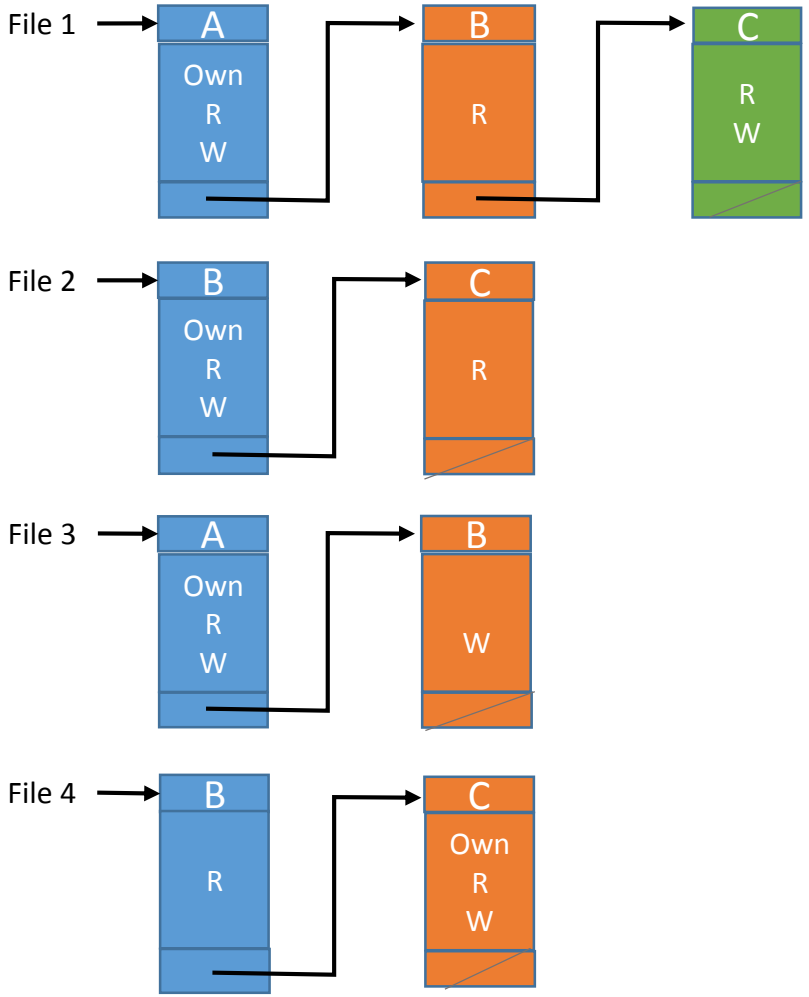
* - copy flag set

Access Control Structures – Decomposing the Access Matrix

SUBJECTS	OBJECTS: File 1	File 2	File 3	File 4
User A	Own Read Write		Own Read Write	
User B	Read	Own Read Write	Write	Read
User C	Read Write	Read		Own Read Write



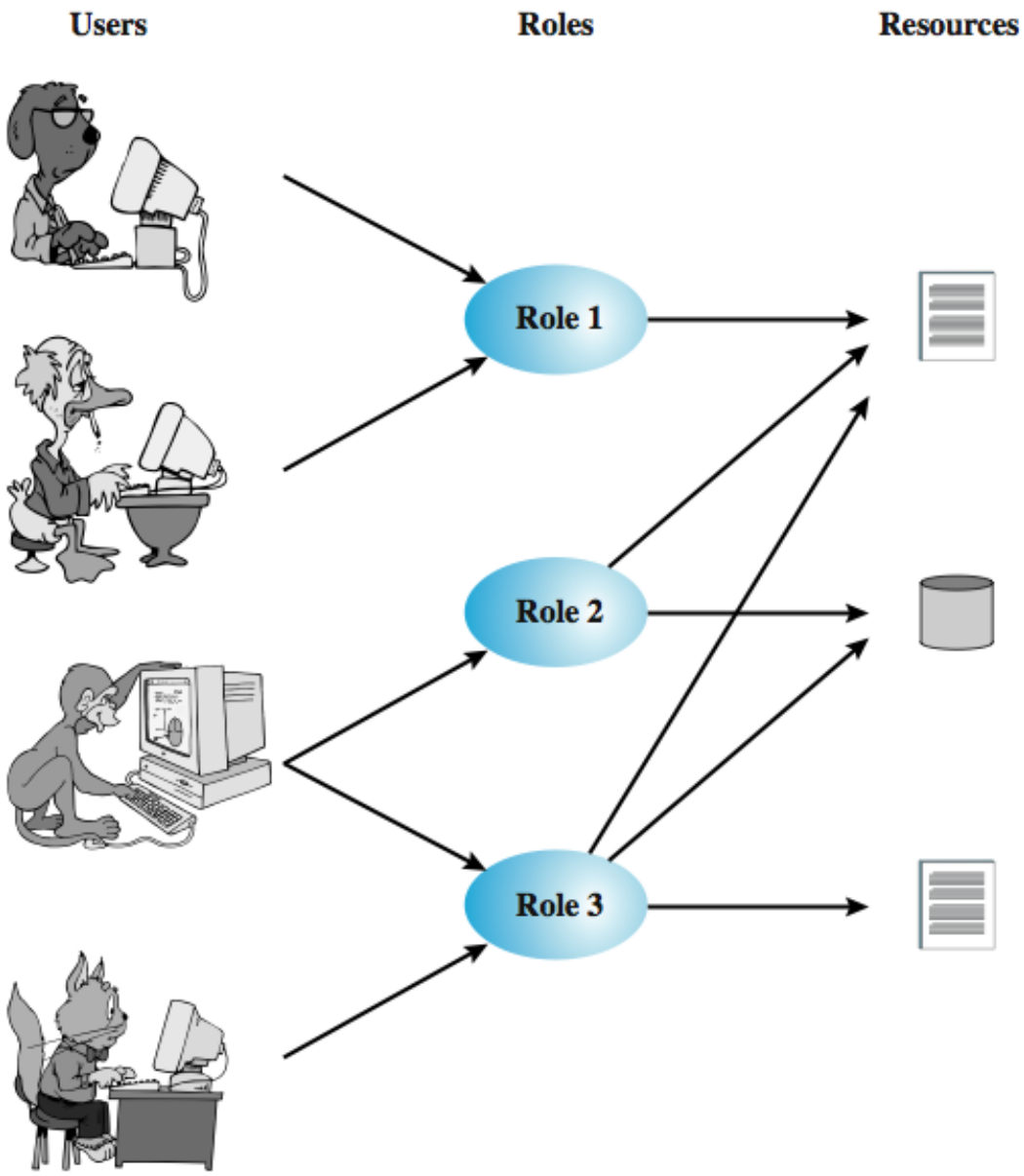
Capability Lists



Access Control Lists (ACLs)

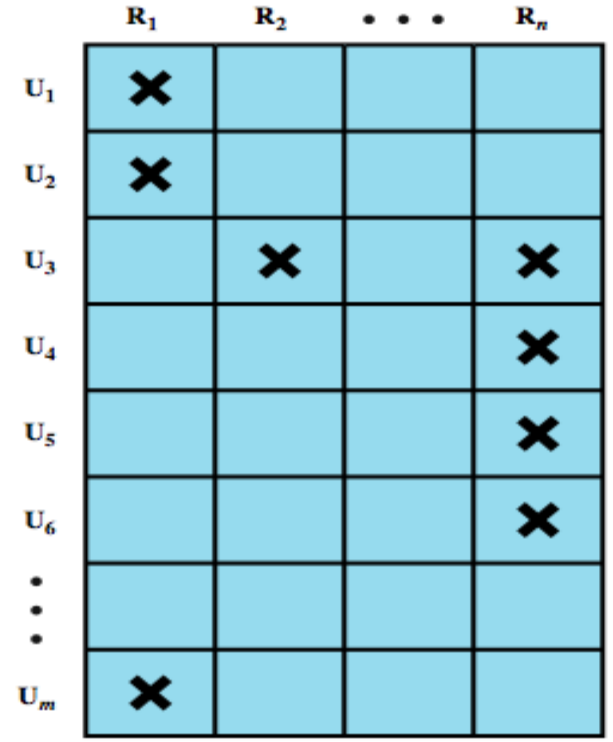
Role-Based Access Model

- Access based on '**role**', not identity
- Many-to-many relationship between users and roles
- Roles often static



Role-Based Access Model

Users-to-roles and roles-to-objects access matrices



	OBJECTS								
ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
R_n			control		write	stop			

Access Control and Cascading Authorizations

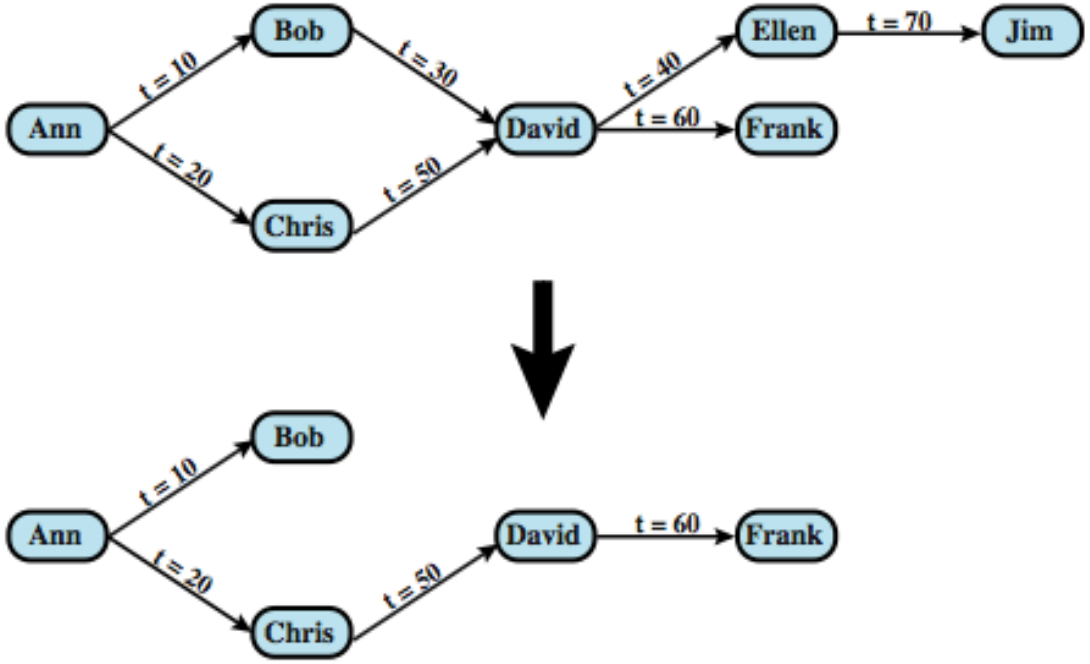
```
GRANT {privileges | role} [ON
table] TO {user | role |
PUBLIC} [IDENTIFIED BY
password] [WITH GRANT OPTION]
```

e.g. GRANT SELECT ON ANY TABLE TO Bob WITH GRANT OPTION

```
REVOKE {privileges | role}
[ON table] FROM {user | role
| PUBLIC}
```

e.g. REVOKE SELECT ON ANY TABLE FROM Bob

WITH GRANT OPTION: whether grantee can grant "GRANT" option to other users



Malware and Attacks

Malware and Attacks

Methods (How is attack launched?)

- Propagation via infected content (e.g. viruses)
- Propagation via vulnerability exploits (e.g. worms, SQL injection, DoS)
- Propagation via social engineering (e.g. trojans)

Goals (What does the payload carry?)

- System corruption (e.g. viruses, logic bombs, DoS)
- Planting attack agents (e.g. bots)
- Information theft (e.g. keyloggers, phishing, SQL injection)
- Stealthing (e.g. backdoors and rootkits)

Malicious software (Malware)

- Programs exploiting system vulnerabilities
  - program fragments that need a host program
    - e.g. viruses, logic bombs, and backdoors
  - independent self-contained programs
    - e.g. worms, bots
  - replicating or not
- Sophisticated threat to computer systems

Virus: *attaches itself to a program*
Worm: *propagates copies of itself to other computers*
Logic bomb: *"explodes" when a condition occurs*
Trojan horse: *fakes/contains additional functionality*
Backdoor (trapdoor): *allows unauthorized access to functionality*
Rootkit: *sophisticated hacker tools to gain root-level access*
Keyloggers: *capture keystrokes*
Spammer and flooder programs: *large volume of unwanted "pkts"*
Zombie/bot: *software on infected computers that launch attack on others (aka bot)*

Virus

- Piece of software that infects programs
  - modifying them to include a copy of the virus
  - so it executes secretly when host program is run
- Specific to operating system and hardware
  - taking advantage of their details and weaknesses
- A typical virus goes through phases of:
  - dormant: *idle*
  - propagation: *copies itself to other program*
  - triggering: *activated to perform functions*
  - execution: *the function is performed*

Virus Structure

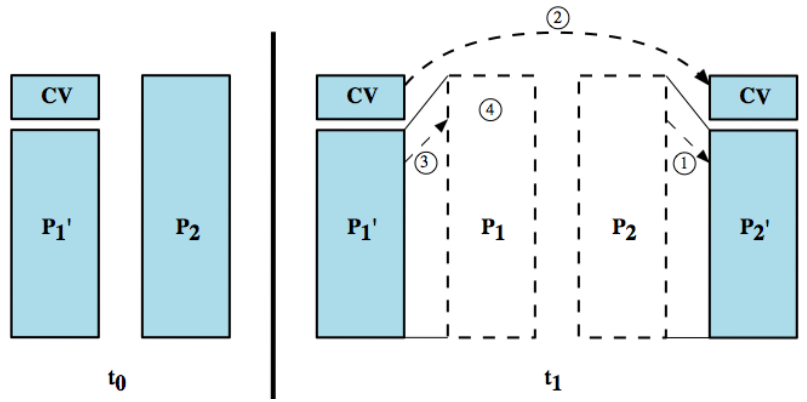
- Components:
  - infection mechanism - enables replication
  - trigger - event that makes payload activate
  - payload - what it does, malicious or benign
- Prepended / post pended / embedded
- When infected program invoked, executes virus code then original program code
- Can block initial infection (difficult)
- Or block propagation (with access control)

Virus Structure

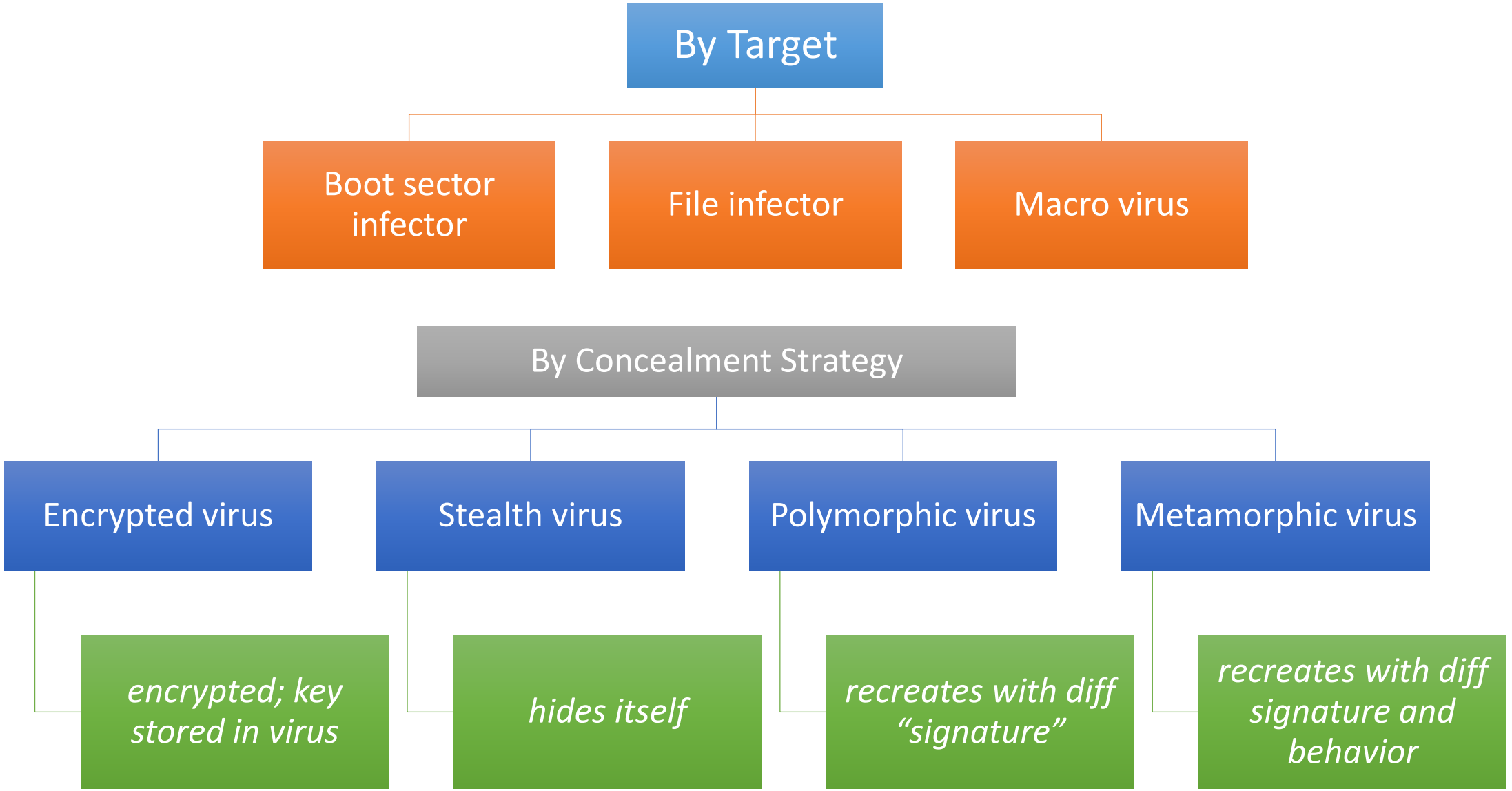
```
    program V :=

{goto main;
    1234567;

    subroutine infect-executable :=
        {loop:
        file := get-random-executable-file;
        if (first-line-of-file = 1234567)
          then goto loop
          else prepend V to file; }

    subroutine do-damage :=
        {whatever damage is to be done}

    subroutine trigger-pulled :=
        {return true if some condition holds}

main:   main-program :=
        {infect-executable;
        if trigger-pulled then do-damage;
        goto next;}

next:

}
```

Compression Virus

```
    program CV :=

{goto main;
    01234567;

    subroutine infect-executable :=
          {loop:
               file := get-random-executable-file;
          if (first-line-of-file = 01234567) then goto loop;
        (1)     compress file;
        (2)     prepend CV to file;
        }

main:   main-program :=
          {if ask-permission then infect-executable;
        (3)     uncompress rest-of-file;
        (4)     run uncompressed file;}
        }
```



Virus Classification



Virus Countermeasures

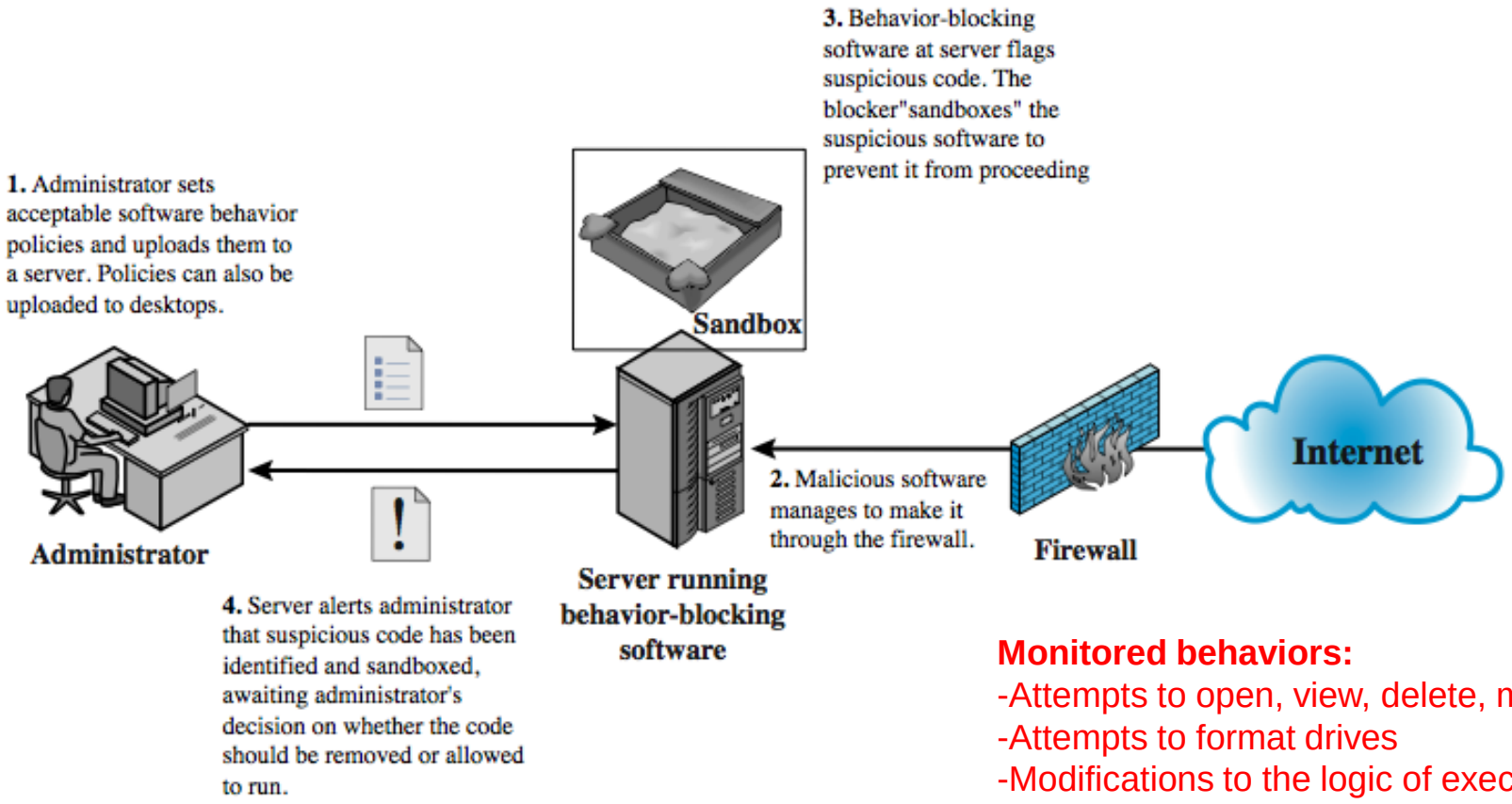
- Prevention - ideal solution but difficult
- Realistically need:
  - detection
  - identification
  - removal
- If detect but can't identify or remove, must discard and replace infected program

Anti-virus Evolution

- Virus & antivirus tech have both evolved
- Early viruses simple code, easily removed
- As viruses become more complex, so must the countermeasures
- Generations
  - first - signature scanners (bit patterns all the same)
  - second – heuristics (integrity checks; checksums)
  - third - identify actions (find by actions they do)
  - fourth - combination packages

Behavior-blocking Software

Integrates with the OS; looks for bad behavior

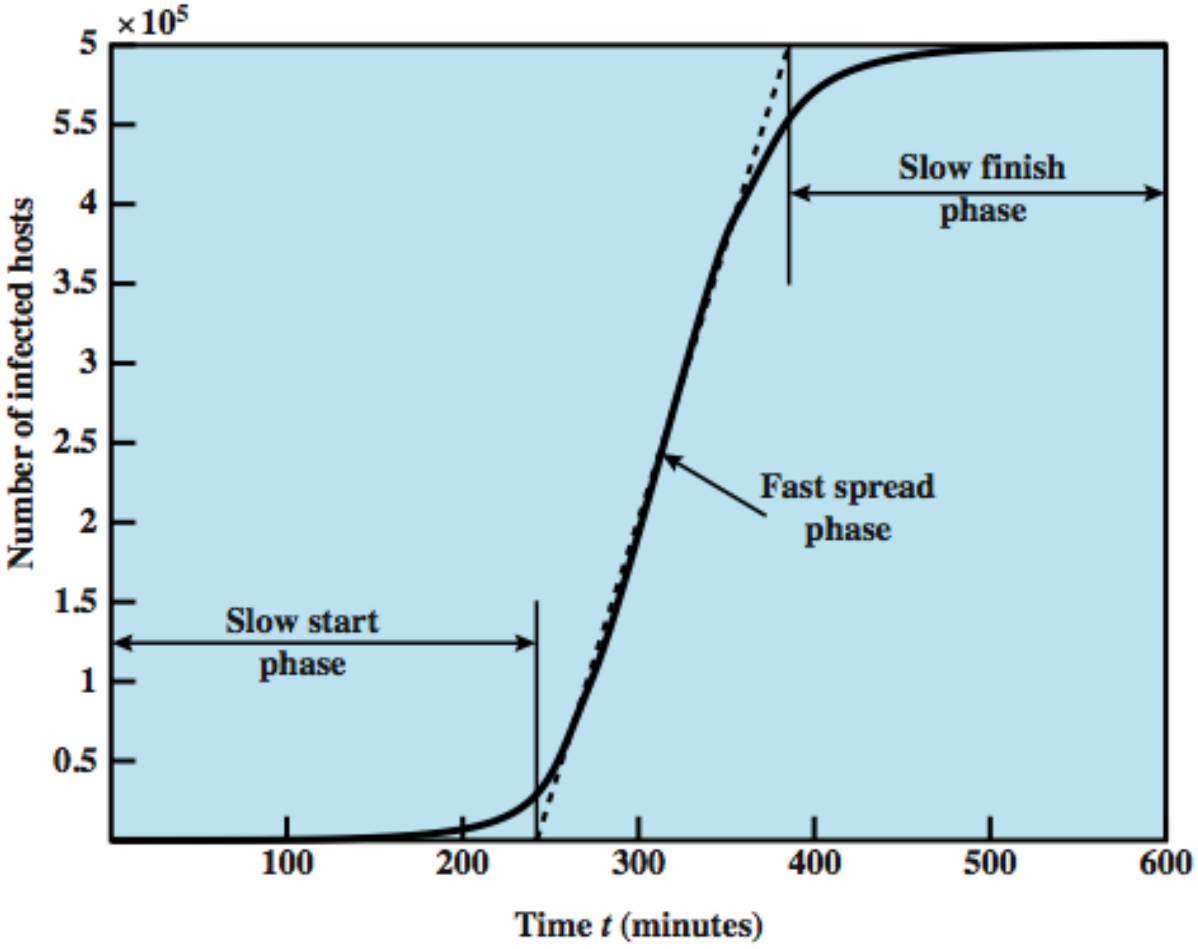


Monitored behaviors:

-Attempts to open, view, delete, modify files
-Attempts to format drives
-Modifications to the logic of executables
-Modifications to critical system settings
-Scripting of emails to send exec contents

Worms

- Replicating program that propagates over net
  - using email, remote exec, remote login
- Has phases like a virus:
  - dormant, propagation, triggering, execution
  - propagation phase: searches for other systems, connects to it, copies self to it and runs



Worm Propagation Model

Infamous Worms

- Morris Worm
  - 1988 on Linux
  - Affected 6,000 computers; cost $10-$100M
  - cracked password file to use login/password to logon to other systems
  - If succeed have remote shell access
- Code Red
  - July 2001 exploiting MS IIS bug
  - probes random IP address, does DDoS attack
  - consumes significant net capacity when active
  - 360,000 servers in 14 hours
- SQL Slammer *(exploited buffer-overflow vulnerability)*
  - early 2003, attacks MS SQL Server
  - compact and very rapid spread
- Mydoom *(100 M infected messages in 36 hours)*
  - mass-mailing e-mail worm that appeared in 2004
  - installed remote access
- Stuxnet
  - targets SCADA systems, responsible for damage to Iran's nuclear program
  - three modules: worm executes payload routines; link file automatically executes propagated copies; rootkit to hide worm's malicious files and processes
  - "One of the great technical blockbusters in malware history"
  - https://www.quora.com/What-is-the-most-sophisticated-piece-of-software-code-ever-written/answer/John-Byrd-2

Worm Countermeasures

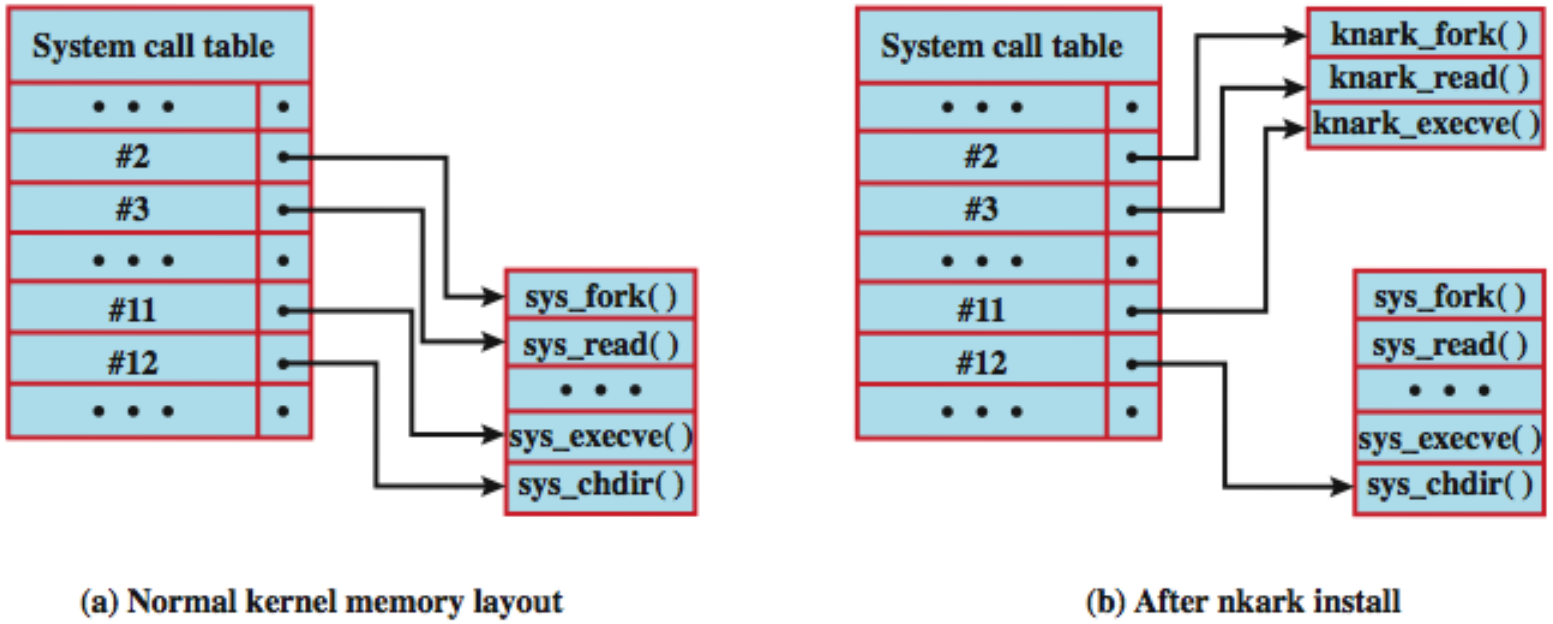
- Overlaps with anti-virus techniques
- Once worm on system A/V can detect
- Worms also cause significant net activity
- Worm defense approaches include:
  - signature-based worm scan filtering
  - filter-based worm containment
  - payload-classification-based worm containment
  - threshold random walk scan detection
  - rate limiting and rate halting

Trojan Examples: rootkits and backdoors

- Trojan horse: looks like a useful tool but contains hidden code
- Can be used to install programs (backdoors) for admin remote access (rootkits)
- Malicious and stealthy changes to host O/S
- May hide existence
  - subverting report mechanisms on processes, files, registry entries etc.
- May be persistent (survives reboot), memory-based, or kernel-based
- Do not rely on vulnerabilities
  - installed via Trojan
  - installed via hackers

Rootkit System Table Mods
A UNIX Example

User API calls refer to a number; the system
maintains a system call table with one entry per number;
each number is used to index to a corresponding system routine



(a) Normal kernel memory layout

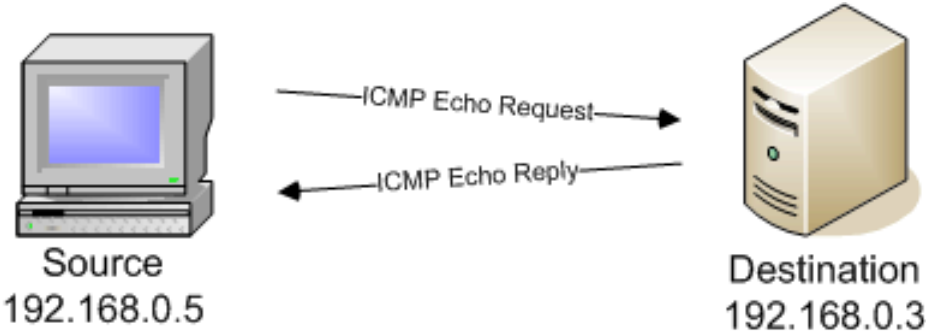
(b) After nkark install

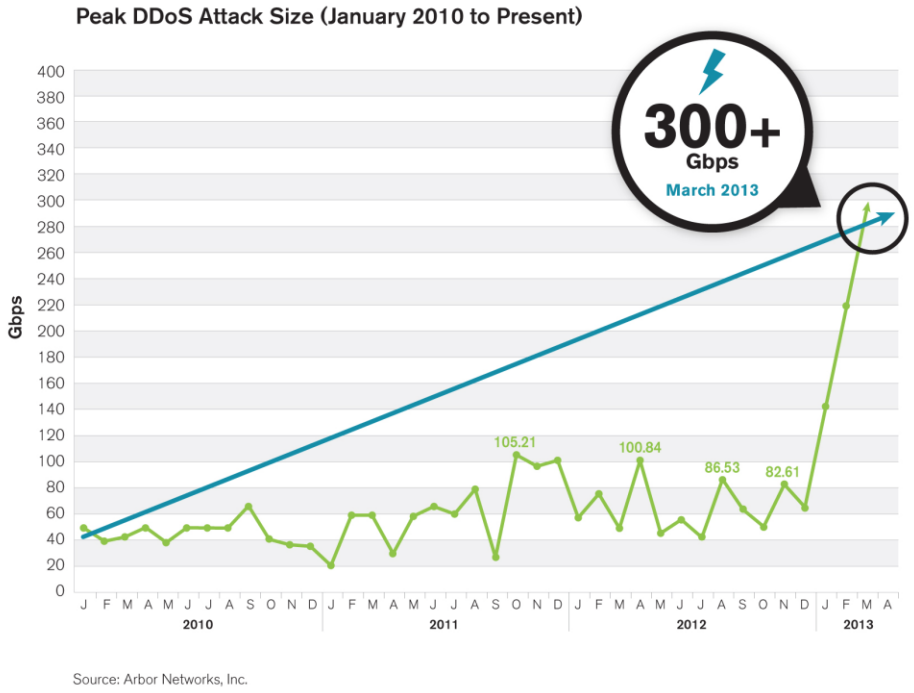
- **Modify the system call table:** modifies syscall addresses
- **Modify system call table targets:** overwrites legitimate system call routines with malicious code
- **Redirect the system call table:** redirects references to the entire system call table to new table in new kernel memory location

rootkit modifies the table and the calls go to
the hackers replacements

Denial of Service Attacks (DoS)

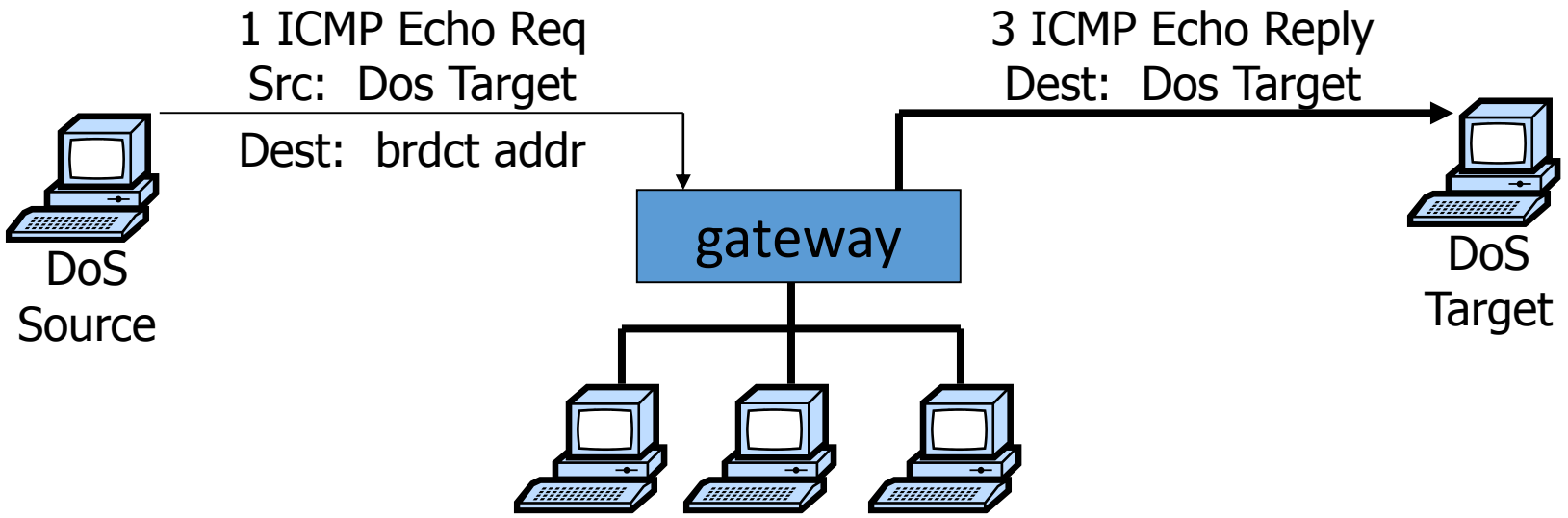
- **Denial of service** (DoS): an action that prevents or impairs the authorized use of networks, systems, or applications
- Attacks (overload or invalid request services that consume significant resources)
  - network bandwidth
  - system resources
  - application resources
- Infamous Examples
  - Ping flooding (e.g. ICMP Echo request flooding)
  - SYN flooding
  - SYN spoofing





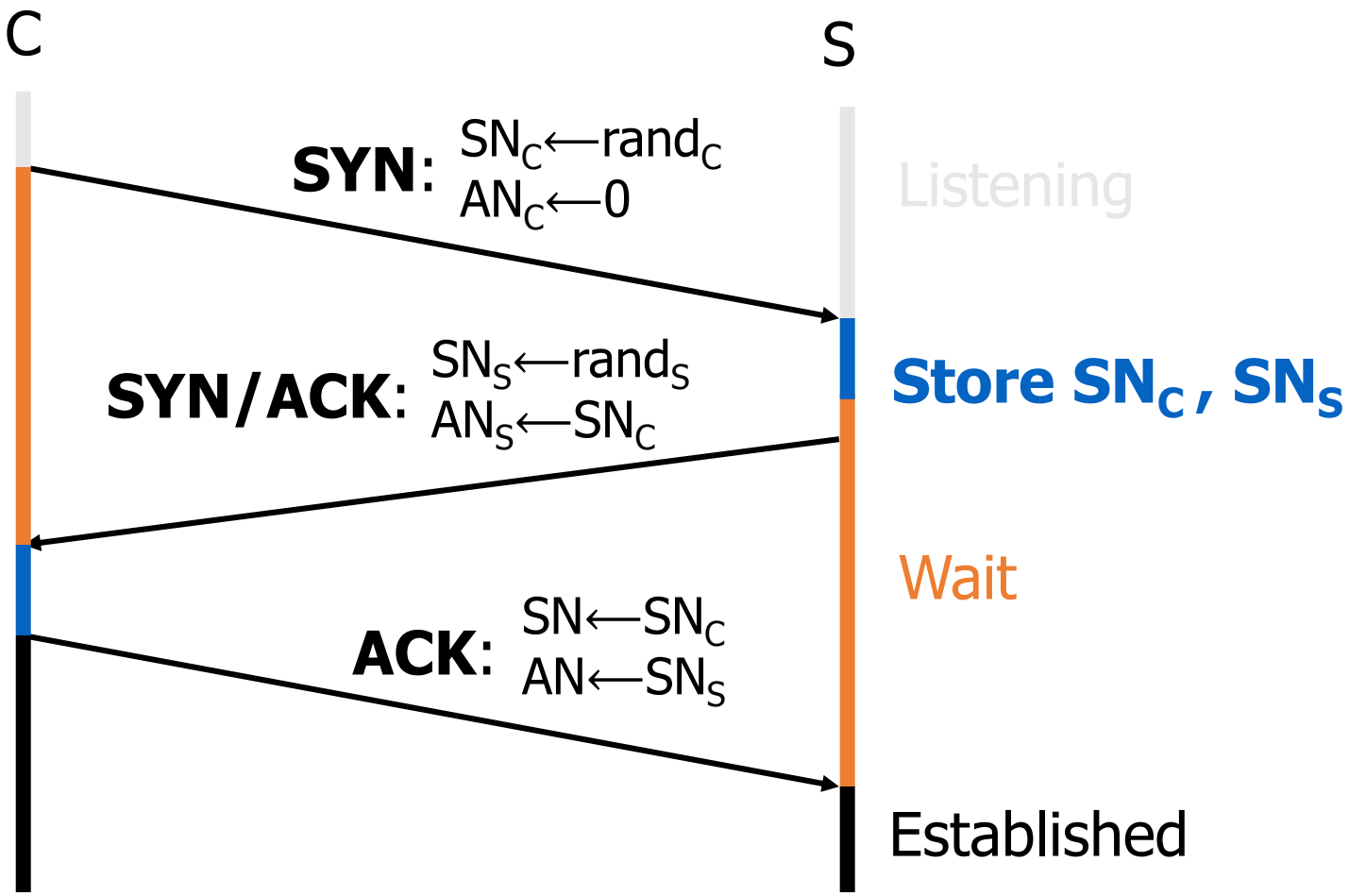
Source: Arbor Networks, Inc.

ICMP Flooding

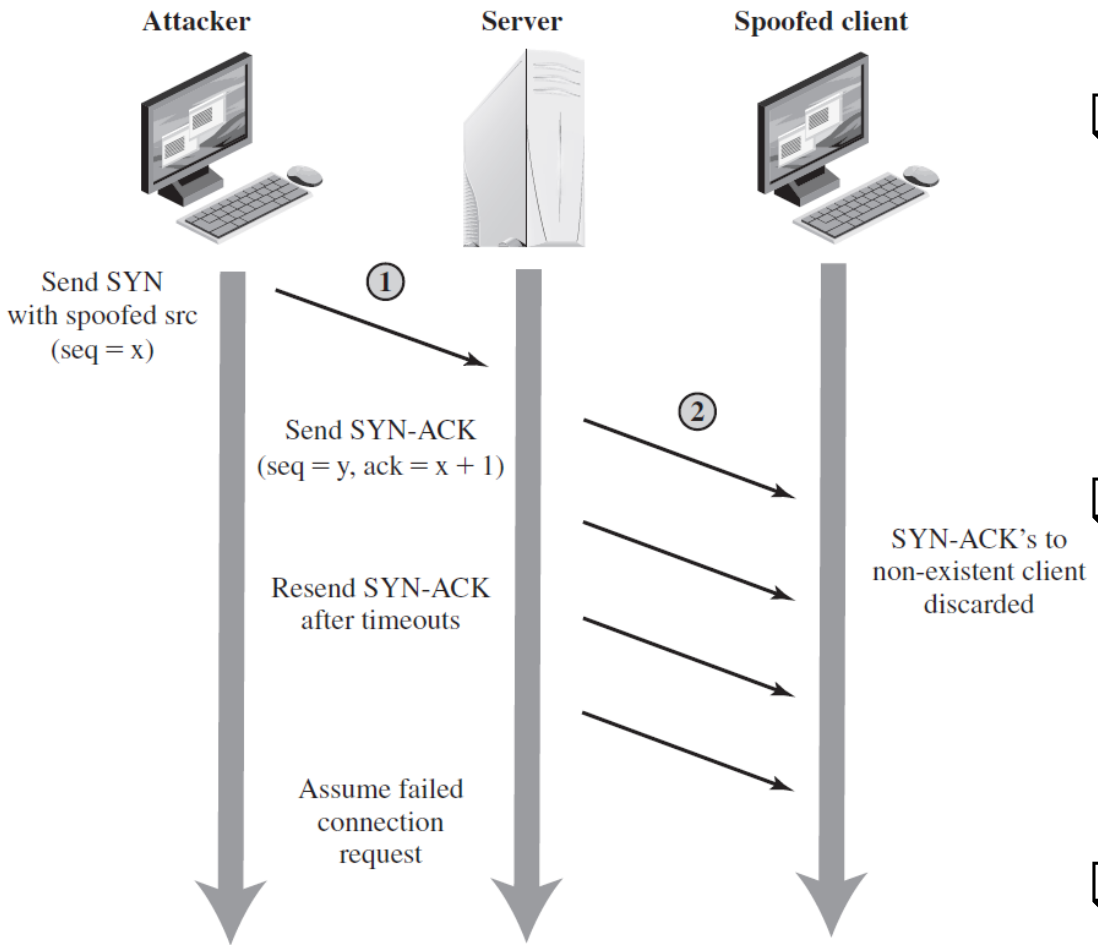


- Source address spoofing → Use forged source addresses
  - via the raw socket interface on operating systems
- Send ping request to broadcast addr (ICMP Echo Req)
- Every host on target network generates a ping reply (ICMP Echo Reply) to victim – more hosts, more flooding!

Review: TCP Handshake

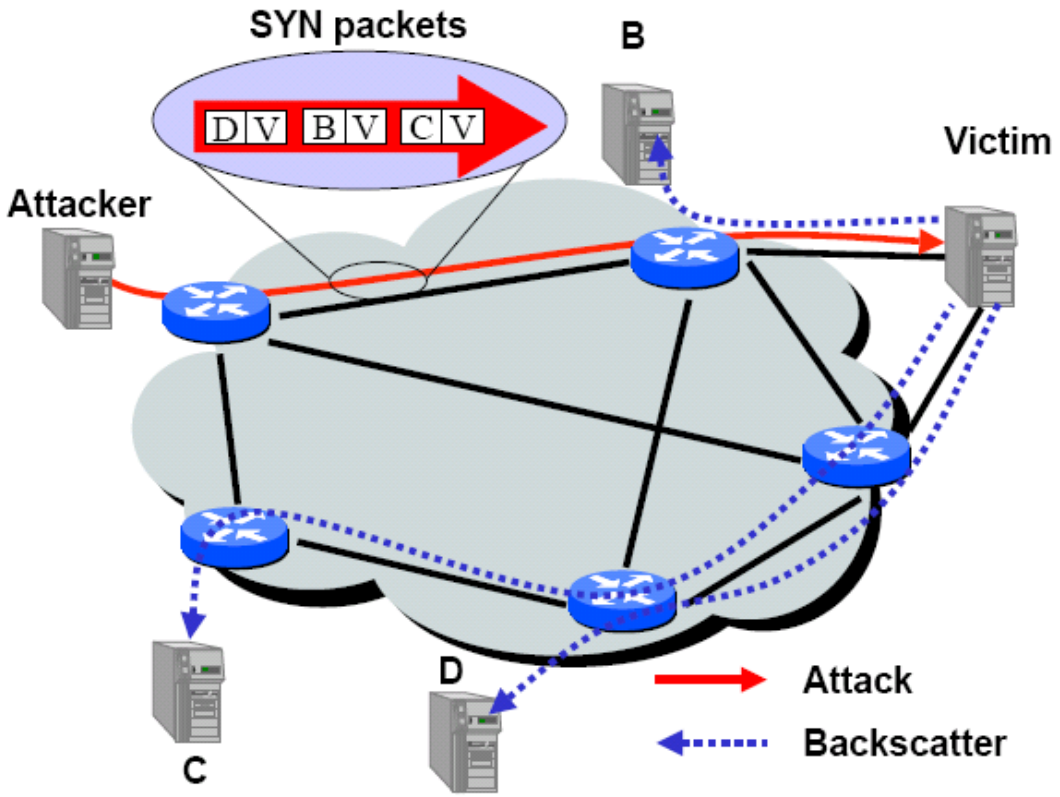


TCP SYN Spoofing



- Attacks ability of a server to respond to future connection requests by overflowing tables that manage them
  - an attack on system resources
- SYN Packets with **random source IP addresses**
  - uses addresses that will not respond to the SYN-ACK with a RST
- Fills up backlog queue on server
- No further connections possible

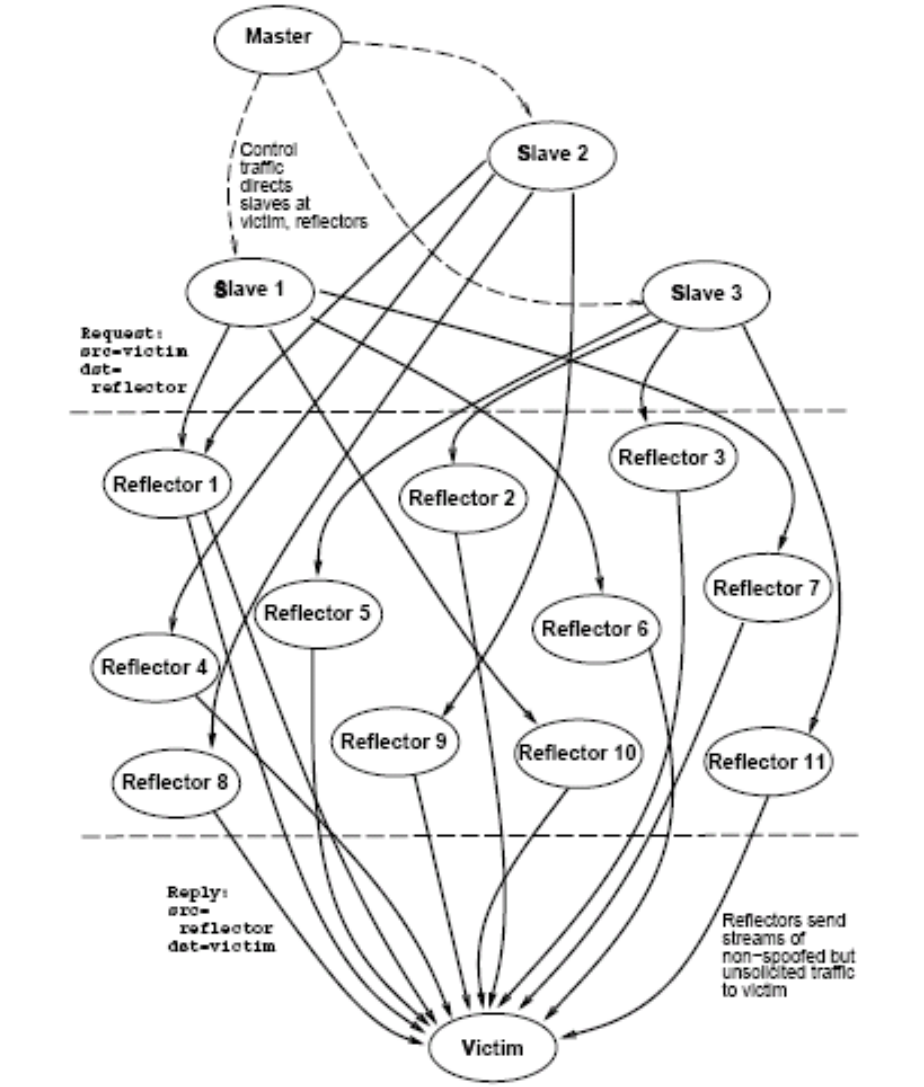
SYN Flooding: backscatter



- SYN with forged source IP ➔ SYN/ACK to random host

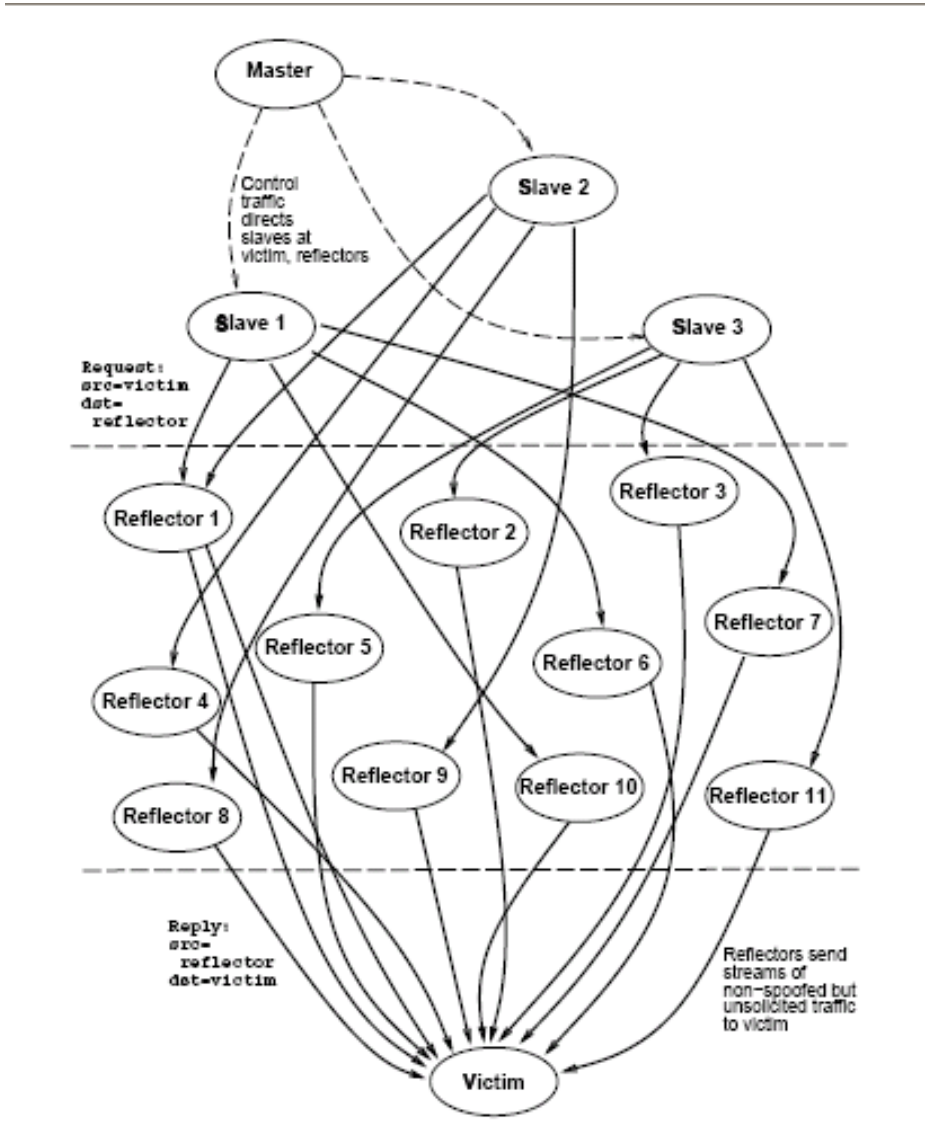
DDoS

- Command bot army to flood specific target
- Saturates network uplink or network router
- Single Master, many bots to generate flood
  - Can use zillions of reflectors to hide bots
  - Kills traceback methods



Reflection and Amplification Attacks

- Attacker sends packet (or packets) to a known intermediary (reflector) with a spoofed source address of the actual target system (victim)
  - Moderate number of request packets generates large volume of response packets to flood link to target system without alerting reflector
- Reflector
  - A network component that responds to packets
  - Response sent to victim (spoofed source IP)
- Examples:
  - DNS Resolvers: UDP 53 with victim.com source
    - At victim: DNS response
  - Web servers: TCP SYN 80 with victim.com source
    - At victim: TCP SYN ACK packet



DoS Countermeasures

- Ingress filtering → ISP only forwards packets with legitimate source IP
  - Implementation problem → ALL ISPs must do this. Requires global trust
    - If 10% of ISPs do not implement ⇒ no defense
    - No incentive for deployment
- Client puzzles → slow down attacker, but checking solution is easy
  - Given challenge C find X such that $\mathbf{LSB_n (SHA\text{-}1(C \| X)) = 0^n}$
  - Assumption: takes expected 2^n time to solve
  - Graphical puzzles (CAPCHA) for application-level DoS attacks
- Syn Cookies → use secret key and data in packet to generate server SN
- Traceback → given set of attack packets, determine path to source
  - change routers to record info in packets
  - Assumption: routers are not compromised, attacker route is relatively stable
- Rate control

SQL Injection Attacks (SQLi)

- One of the most prevalent and dangerous network-based security threats
- Sends malicious SQL commands to the database server
- Depending on the environment SQL injection can also be exploited to:
  - Modify or delete data
  - Execute arbitrary operating system commands
  - Launch denial-of-service (DoS) attacks
- The SQLi attack typically works by prematurely terminating a text string and appending a new command

SQL Injection Attacks (SQLi)

Tautology: injects code in one or more conditional statements so they always evaluate to true

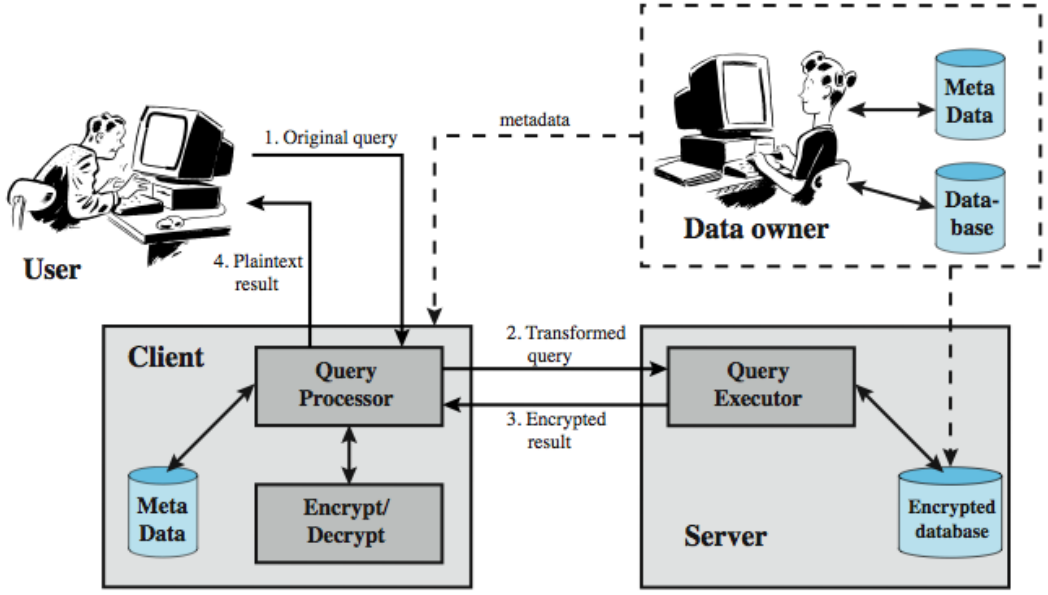
End-of-line comment: After injecting code into field, legitimate code that follows is nullified through use of end of line comments

```
var Shipcity;
ShipCity = Request.form (“ShipCity”);
var sql = “SELECT * FROM OrdersTable WHERE
ShipCity = ‘” + ShipCity + “’”;

Attacker enters: Redmond’; DROP table
OrdersTable--

SELECT * FROM OrdersTable WHERE ShipCity =
‘Redmond’; DROP table OrdersTable--
```

```
$query= “SELECT info FROM user WHERE name =
‘$_GET[“name”]’ AND pwd = ‘GET[“pwd”]’”;

Attacker enters: ‘ OR 1=1 –-

SELECT info FROM users WHERE name = ‘ ‘ OR
1=1 -- AND pwd = ‘ ‘
```

SQL Injection Countermeasures

- Defensive coding: stronger data validation
- Detection
  - Signature based
  - Anomaly based
  - Code analysis
- Runtime prevention: Check queries at runtime to see if they conform to a model of expected queries
  - e.g. parametrized insertion
- Database encryption
- Database access control



Can encrypt

- entire database - very inflexible and inefficient
- individual fields - simple but inflexible
- records (rows) or columns (attributes) - best
  - also need attribute indexes to help data retrieval