Project 2: 23 days left

Deception-Based Cyber Defense: Moving Target Defense

CS 459/559: Science of Cyber Security
20th Lecture

Instructor:

Guanhua Yan

Agenda

- ~~Quiz 1: September 29 (closed book)~~
- ~~Project 1 (offense): October 10~~
- Course wrapup & presentation lottery: November 10
- Quiz 2: November 12
- Presentations: 11/17, 11/19, 11/24, 12/1, 12/3
- CTF competition: November 26
- Project 2 (defense): December 5
- Final report: December 15

CTF leaderboard

	User Name	Successful Attack	Score
1	sandworm	Buffer Overflow, DNS Tunneling, Known Plaintext Attack, Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	120
2	jeff	Buffer Overflow, DNS Tunneling, Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	110
3	slee	Buffer Overflow, DNS Tunneling, Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	110
4	JamesRatanDukkipati	Buffer Overflow, DNS Tunneling, Network Reconnaissance, Program Wrapper(partial), Reverse Proxy, SQL Injection	100
5	azeng8	DNS Tunneling, Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	100
6	dchaganti	DNS Tunneling, Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	100
7	Sandeep	DNS Tunneling, Network Reconnaissance, Reverse Proxy, SQL Injection	80
8	Srimunagala	DNS Tunneling, Network Reconnaissance, Reverse Proxy, SQL Injection	80
9	himan	Buffer Overflow, Network Reconnaissance, Program Wrapper, SQL Injection	80
10	cabbineni	Program Wrapper, Reverse Proxy, SQL Injection, Tiny Shell Exploit	80
11	asatishkumar	Network Reconnaissance, Program Wrapper, Reverse Proxy, SQL Injection	80

Outline

- What is moving target defense (MTD)?
- IP obfuscation
- OS & service obfuscation
- Dynamic system
- Dynamic software

Motivation

- “Just as water remains no constant shape, in warfare there are no constant conditions.” – By Sun Tzu.

Moving target defense (MTD)

- In cyber defense, a dynamic, constantly evolving attack surface for the protected system is extremely valuable to retain a resilient security posture.
- MTD techniques seek to randomize system components to:
  - Reduce the likelihood of a successful attack
  - Increase system dynamics to reduce the lifetime of an attack
  - Diversify otherwise homogeneous systems to limit the damage of a large-scale attack.
- MTD intensifies uncertainty and workload for attackers by making the protected system less static, less deterministic, and less homogeneous.

MTD strategies

- In the **external reconnaissance** phase, attackers have to gain necessary knowledge about the target system before they can move on along the kill chain.
  - IP obfuscation
  - OS/service obfuscation

- The **exploitation** attack phase may be guarded against by various dynamic system and software techniques:
  - Dynamic system
  - Dynamic software

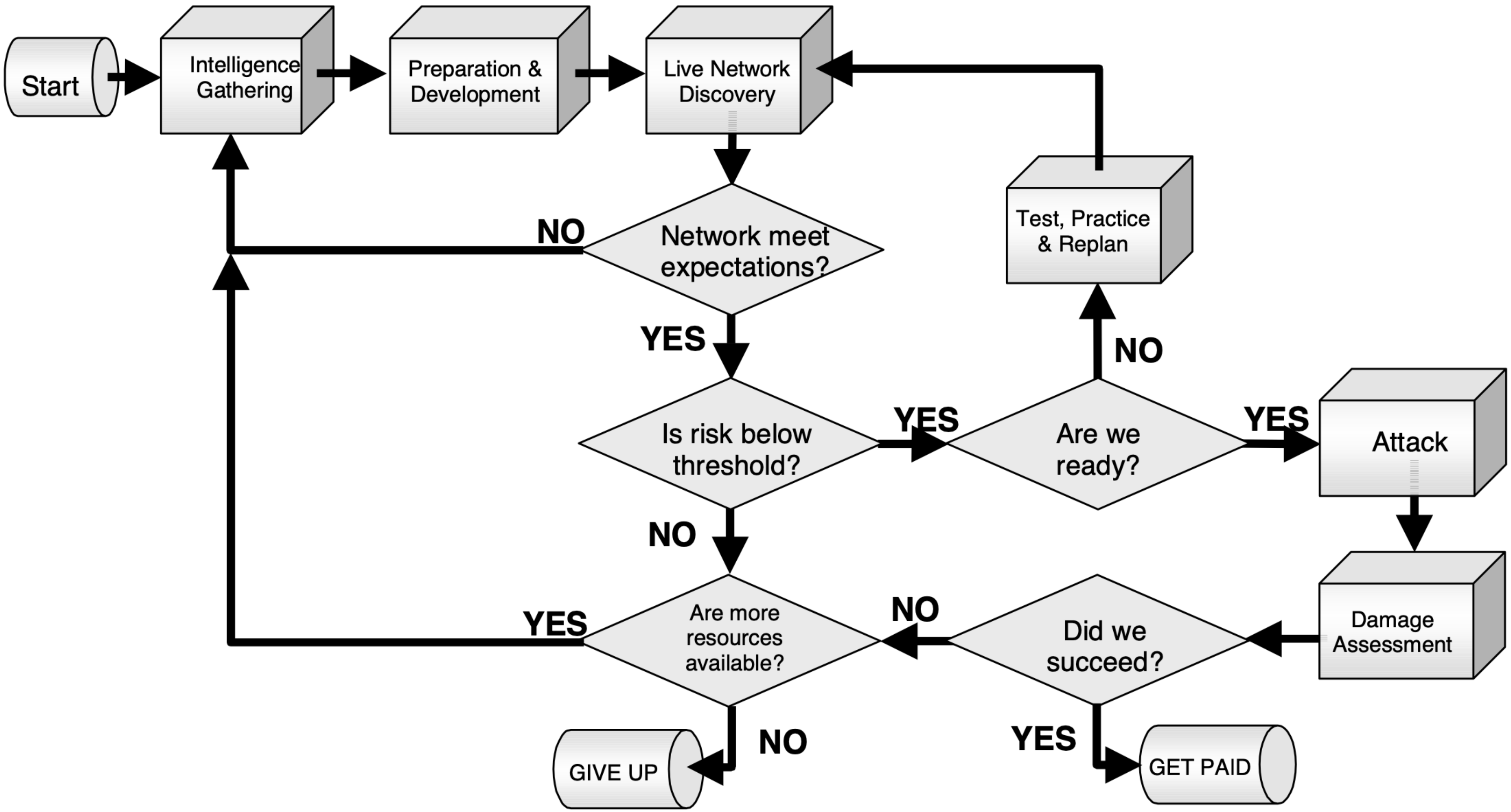
IP Obfuscation

IP obfuscation

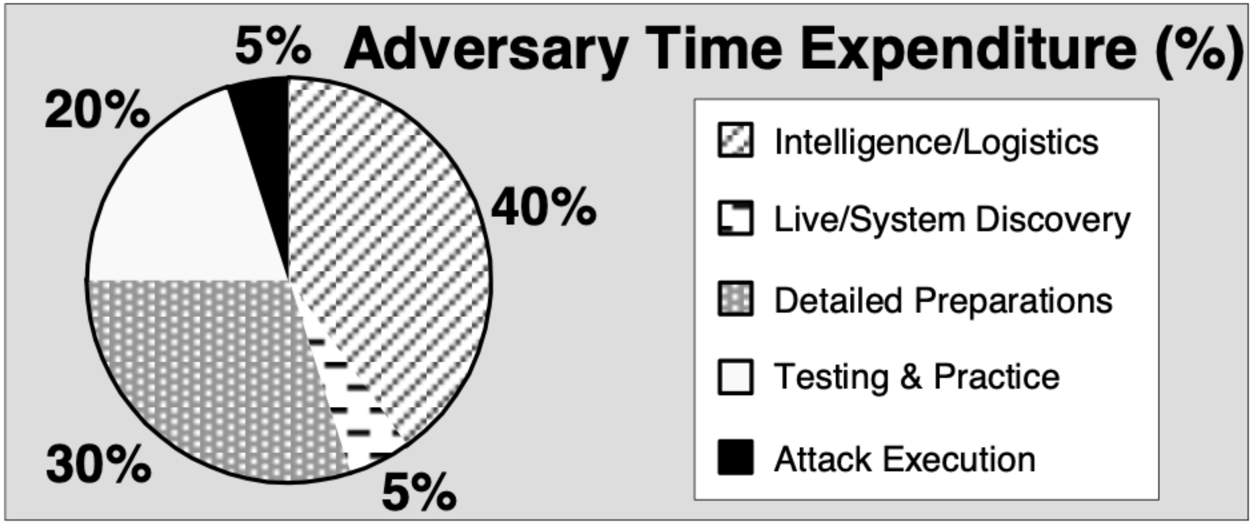
- Goal: Prevent attackers from tracing hosts in the target network based on IP addresses.

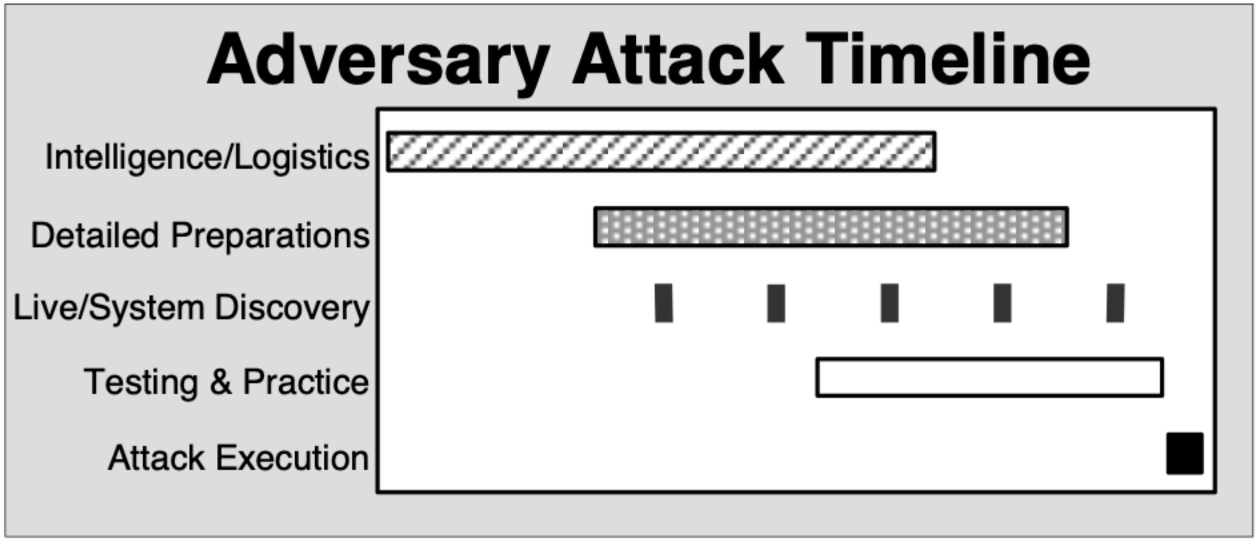
- Representative solutions:
  - **DyNAT: dynamic network address translation**
  - **NASR: network address space randomization**

DyNAT: Attack process



DyNAT: Adversary work distribution



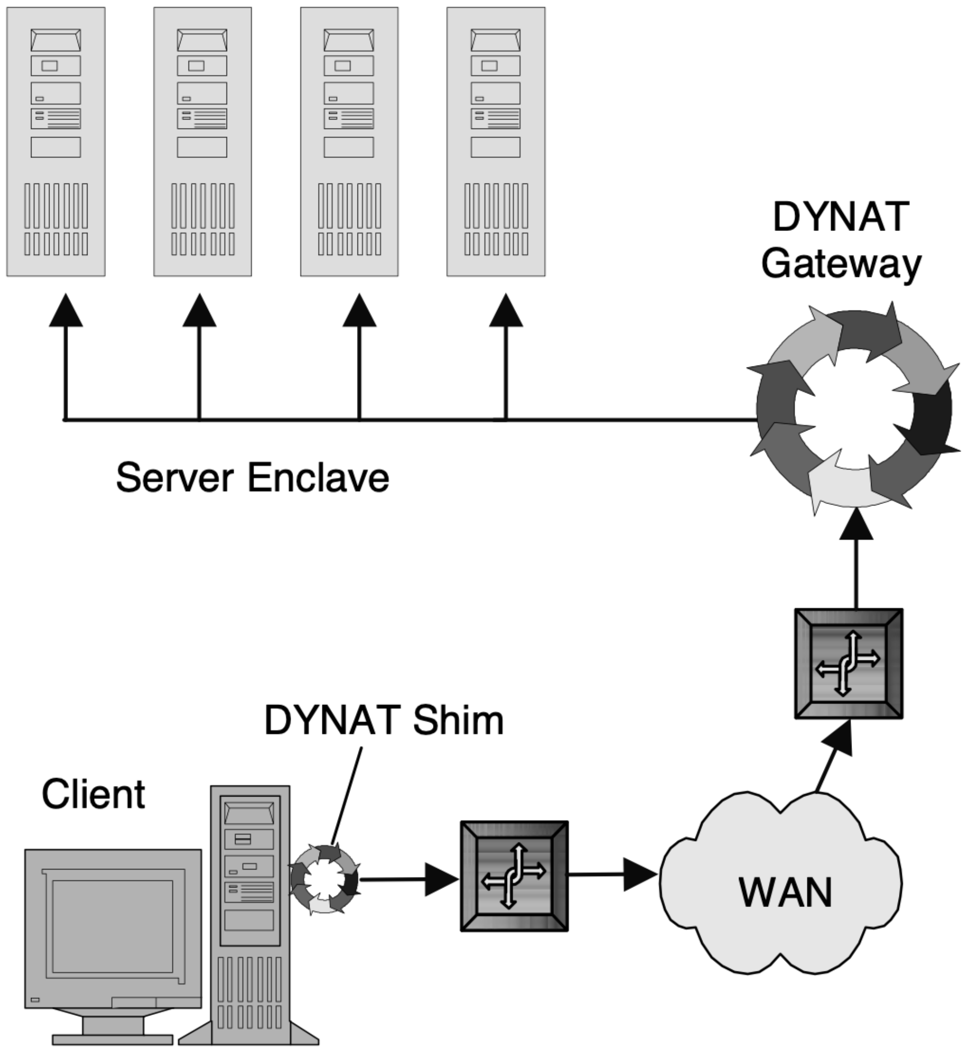


DyNAT: Rationale

- Dynamic network modifications make it harder for the adversary to map the network. By dynamically modifying logical network locations, we limit the time value of intelligence gathered (sniffed) by the adversary. If an attack is not executed before the network changes, the attacker risks failure and apprehension.

- By causing adversaries to repeat the intelligence gathering and development phases of the attack, we cause the adversaries to expend additional time and resources to reach their goals. If we can keep them in this cycle, they will never attain sufficient confidence in their intelligence or their attack methodology and thus never launch their attacks.

DyNAT: Architecture



DyNAT: Methodology

- DYNAT operates by obfuscating host identity information in TCP/IP packet headers when packets enter public parts of the network.
- Addressing information originating from a sending client host is translated in the datagram header by the DYNAT shim prior to routing to the receiving server enclave.
- The translation algorithm depends on a preestablished keying parameter that varies with time. The receiver, which is a server gateway, reverses the translation in the header fields to obtain the true host identity information.
- The datagram, with the original identity information, is passed into the server enclave for normal processing.

DyNAT: Destination address and port

<table>
<tr><td rowspan="6">I
P</td><td>Ver.</td><td>IHL</td><td>TOS</td><td colspan="3">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Bits</td><td colspan="2">Frag Offs</td></tr>
<tr><td colspan="2">Time to Live</td><td>Protocol</td><td colspan="3">Header Checksum</td></tr>
<tr><td colspan="5">Source Network Address</td><td>Src Host Addr</td></tr>
<tr><td colspan="5">Dest Network Address</td><td>Dest Host Addr</td></tr>
<tr><td colspan="6">Options</td></tr>
</table>

<table>
<tr><td rowspan="6">T
C
P</td><td colspan="2">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">Acknowledgement Number</td></tr>
<tr><td>Offset</td><td>Reserved</td><td>(Bits)</td><td>Window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urg Ptr</td></tr>
<tr><td colspan="4">Options</td></tr>
</table>

DyNAT: Address translation

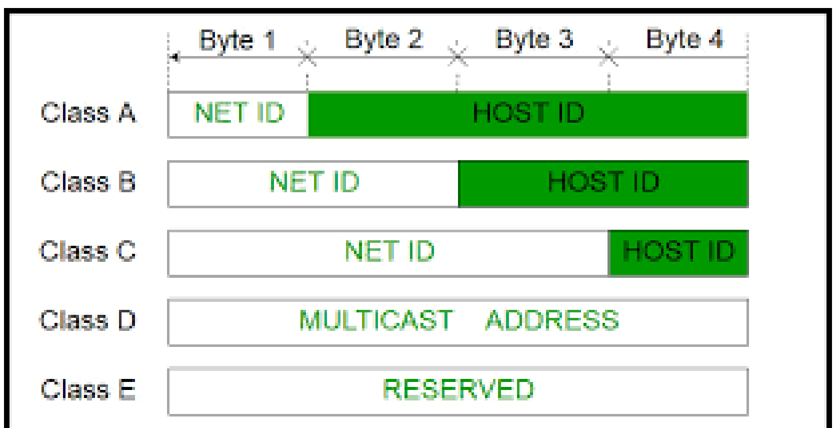
- Only the host portion of the destination address is translated, not the network address.

DyNAT: Address translation

- Only **the host portion of the destination address** is translated, not the network address.

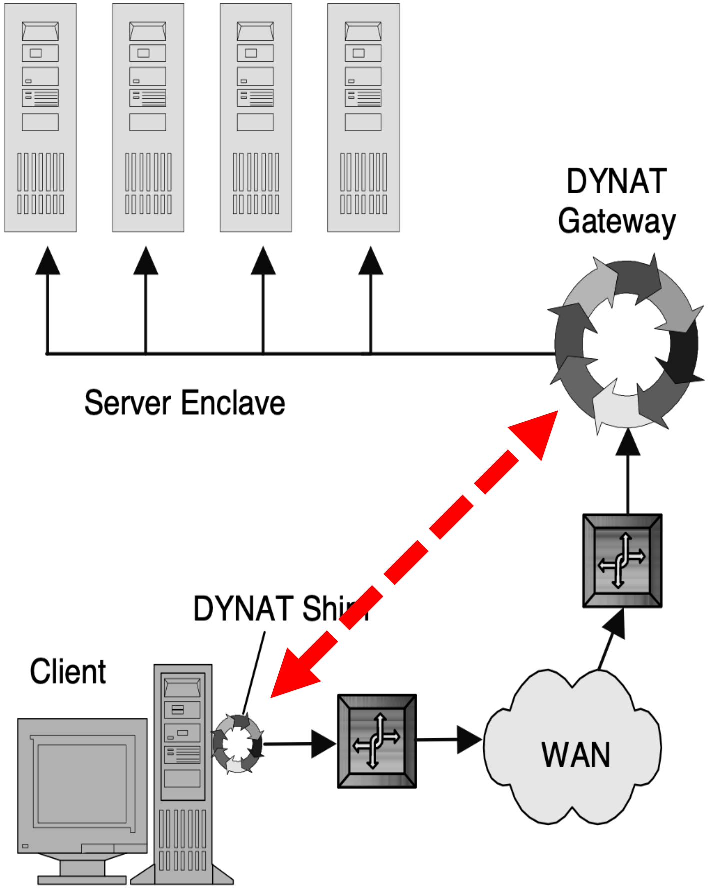
- Thus, the packet can be routed normally.

- The number of bits that are translated is a function of the class of IP address in use.

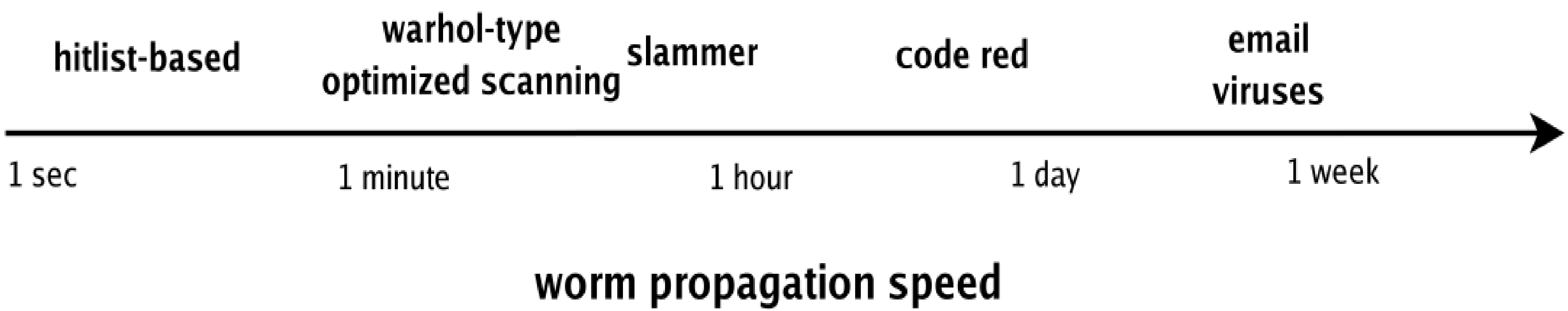


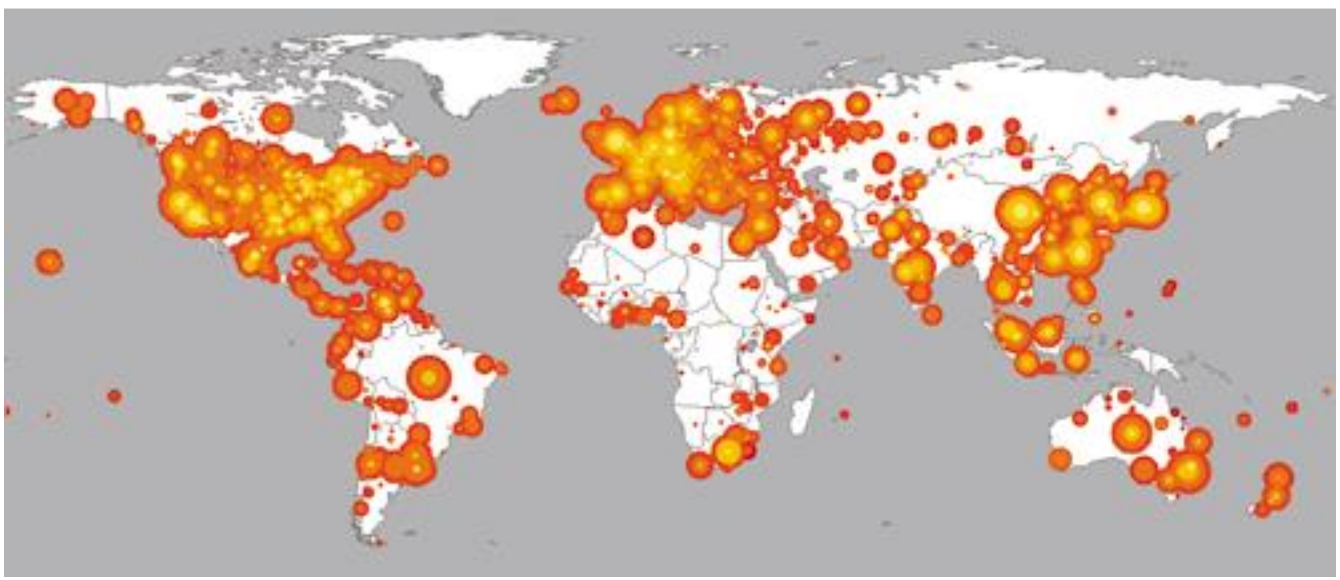
DyNAT: Address translation

- Translation is performed by a cryptographic algorithm.
  - The client (i.e., the originating DYNAT translator host) and the server (i.e., the recipient DYNAT translator host) are configured with an initial secret seed value.
  - A time-based mechanism was used to periodically change the secret and thus change the translation results, making it difficult for the adversary to create and maintain a map of the network.



NASR: Hitlists worm & key idea

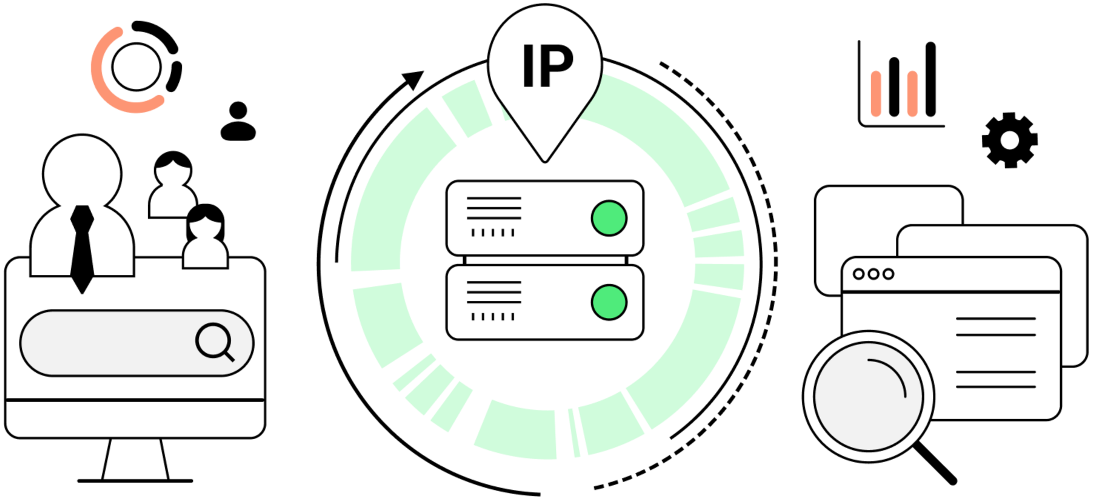




CAIDA Analysis of Code-Red Worm (2001)

NASR: Key idea

- Adapting dynamic network address allocation service (e.g., DHCP) to force more frequent address changes.



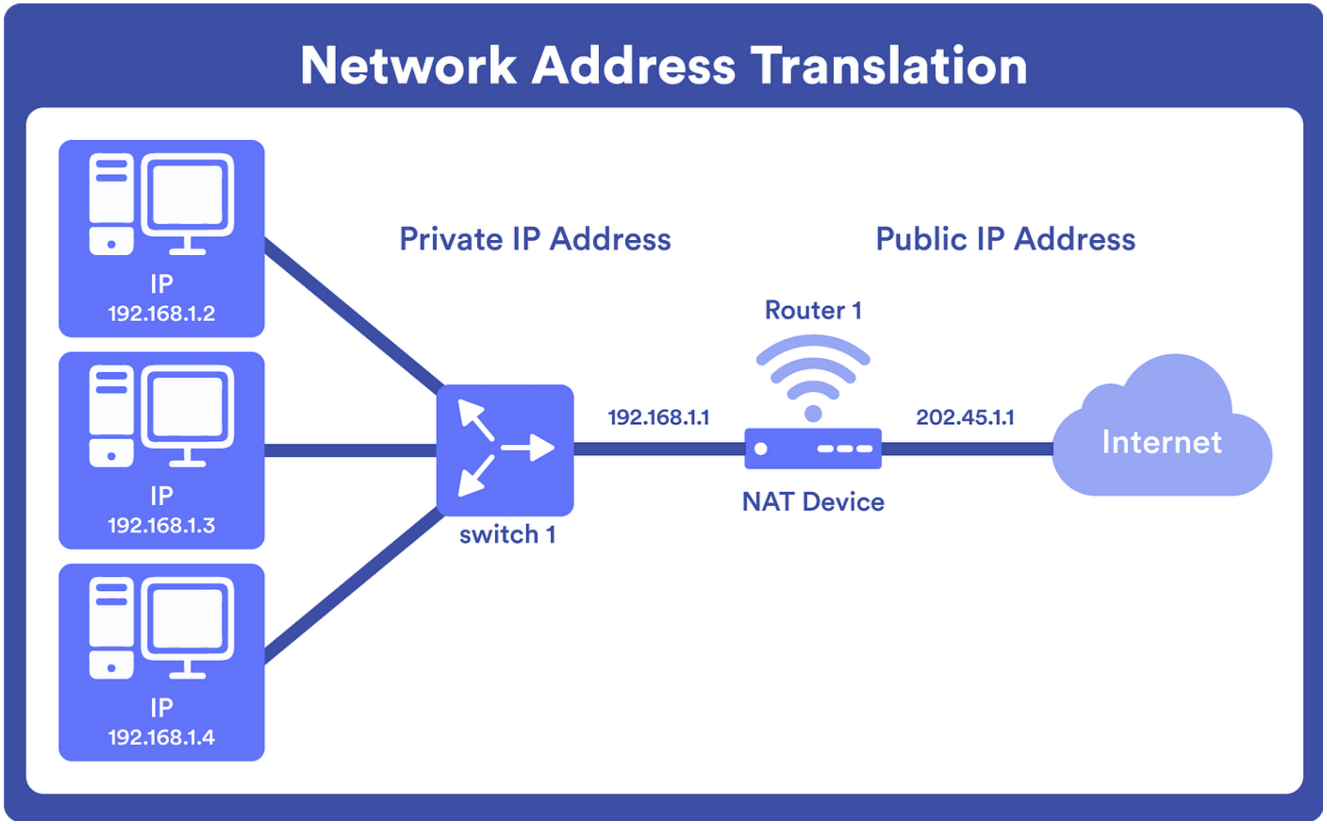
NASR: Scope

- Random assignment of an address from a global IP address space pool is not practical (routing, administrative cost, global coordination).

- It seems that a reasonable strategy would be to provide NASR at the subnet-level.

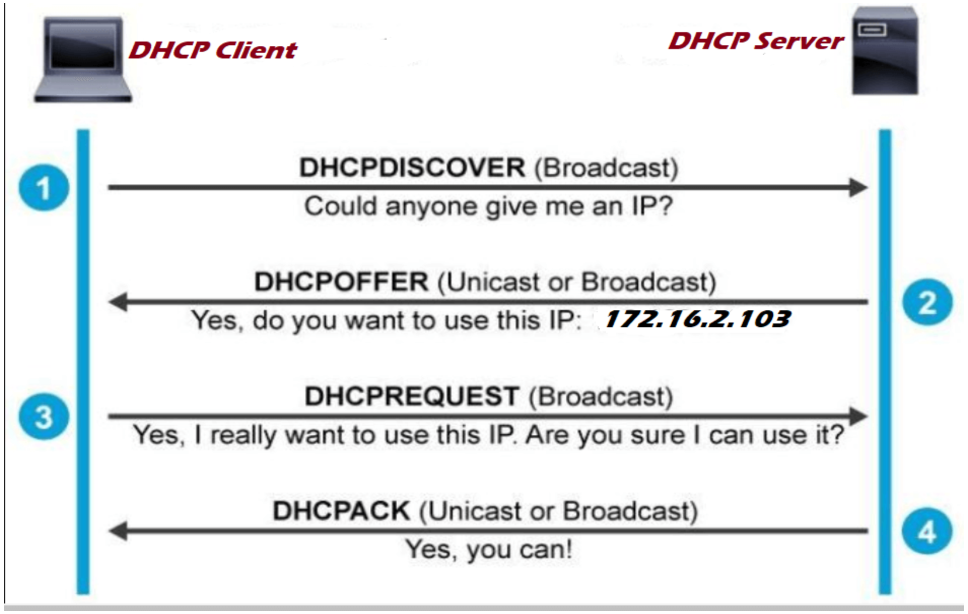
NASR: Scope

- It is also obvious that it is pointless to implement NASR behind NATs, as the internal addresses have no global significance.



NASR: DHCP

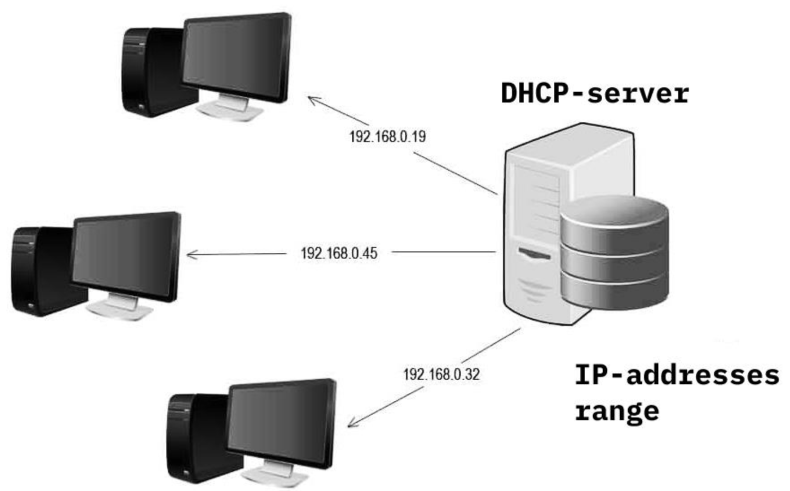
- DHCP: Dynamic Host Configuration Protocol



- The DHCPRequest packet includes the request of network parameters from the DHCP server.
- The DHCPACK packet includes the lease duration and other configuration information.

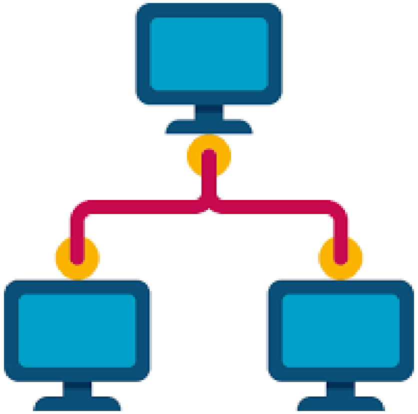
NASR: Implementation

- A basic form of NASR can be implemented by configuring the DHCP server to expire DHCP leases at intervals suitable for effective randomization.
- Forcing addresses changes even when a host requests to renew the lease before it expires requires some minor modifications to the DHCP server.



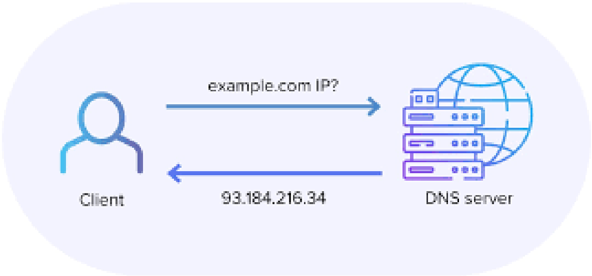
NASR: Implementation

- An advanced NASR-enabled DHCP server is built based on an open-source DHCP implementation.
- To minimize the "collateral damage" caused by address changes two modules are used:
  - The **activity monitoring** module keeps track of **open connections** for each host with the goal of avoiding address changes for hosts whose services could be disrupted.
  - **Service fingerprinting** examines **traffic** on the network and attempts to identify what services are running on each host.

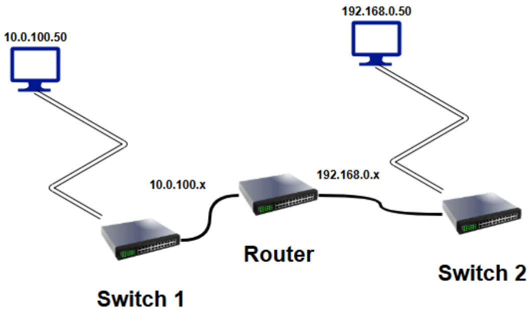


NASR: Static addressing

- Some nodes cannot change addresses because their addresses have first-class transport- and application-level semantics.
  - For instance, DNS server addresses are usually hardcoded in system configurations.

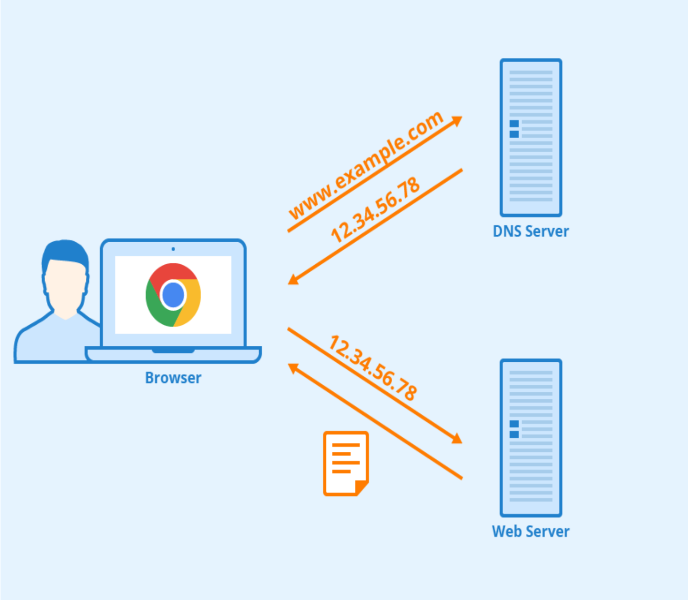


  - Similar constraints hold for routers.



NASR: DNS updates

- For services referenced through the DNS name, such as email, FTP and Web servers, implementing NASR requires the DNS name to accurately reflect the current IP address of the host.
  - The DNS time-to-live timers need to be set low enough so that remote clients and name servers do not cache stale data when an address is changed.
- The NASR mechanism also needs to interact with the DNS server to keep the address records up to date.



OS and Service Obfuscation

OS and service obfuscation

- **Goals: obfuscate OS and service information**

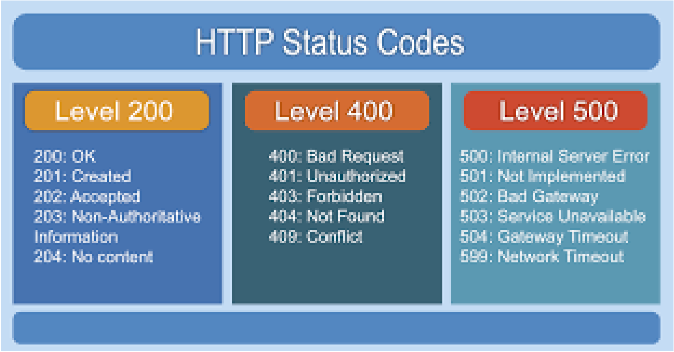
- **Examples:**
  - MTD against software version identification
  - MTD against OS fingerprinting

Reconnaissance: software version

- To exploit a known software vulnerability through the network, an attacker first needs to identify the vulnerable service and its version, unless the vulnerability is a 0-day.
- Upon confirming that the version is vulnerable, the exploit can be deployed successfully.
- If the version cannot not be identified, then the attack surface widens significantly, and the attacker practically would be shooting in the dark, if he was attempting to exploit a known vulnerability.

Version identification for HTTP service

- An HTTP server sends the httpd daemon service version in the HTTP header of an HTTP 200 OK response to an HTTP GET.
- If the version in the response is not a known version (i.e., obfuscation is happening), an attacker or a tool could generate **requests for an unknown resource** and see the httpd version in the HTTP 404 Not Found response.
- Alternatively, the attacker can **generate an invalid SIP message in HTTP** and retrieve the version from the HMTL code in the HTTP 400 response from the server.



MTD against version identification

- All cases where the httpd version is included in HTTP server responses will be overridden with a bogus service version.

- Of course, such applications could have a caveat of breaking if the service version is necessary for the service transactions.

OS fingerprinting

- Although modern OSes try to generate truly random responses, TCP sequence numbers and TCP, ICMP and UDP payloads in response to certain packets can be used sometimes to find what OS the target is running.

- A Linux flavor generates randomized TCP sequence numbers in a specific way which is different from others.

- The payload pattern in TCP and UDP packets generated by a host could reveal information for the host's OS.

MTD against OS fingerprinting

Algorithm 2 MTD against OS fingerprinting

```
while (new TCP packet p destined to target is received) do
  if (p is illegitimate traffic) then
    if (p has TCP SYN set) then
      s ← random 32-bit number
      respond with TCP SYN-ACK and s as the seq#
    else
      generate random payload and respond
    end if
  end if
end while
```

Dynamic System

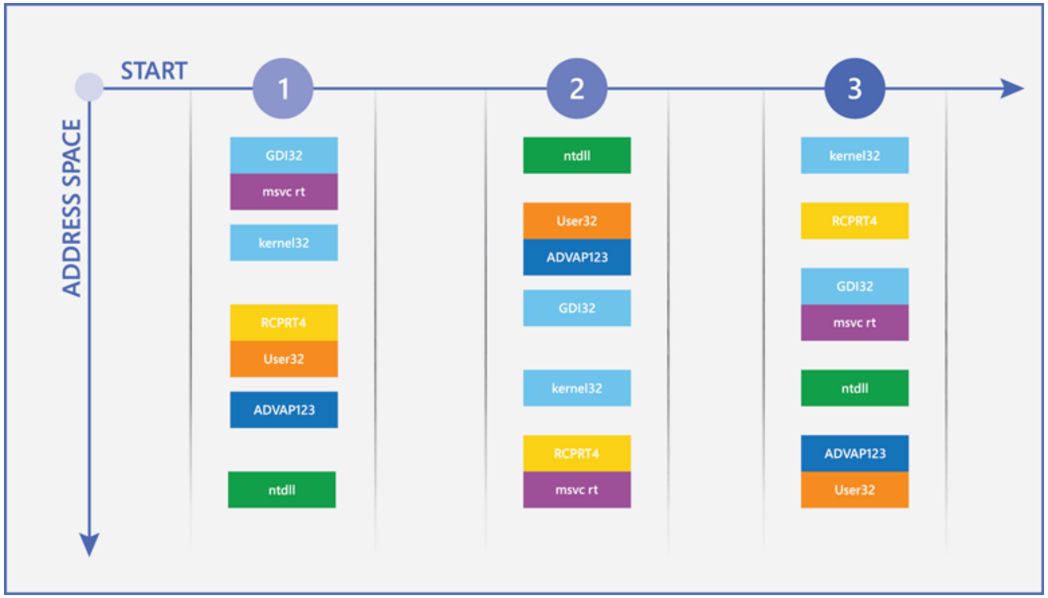
Dynamic system

- **Key idea: MTD at the system level**

- **Examples:**
  - Address Space Layout Randomization
  - Instruction set randomization
  - Multiple rotational OS
  - TALENT

ASLR

- **ASLR (Address Space Layout Randomization)** hinders the exploitation of memory corruption vulnerabilities by randomizing memory addresses of a loaded software.



Microsoft OS

ISR

- The instruction set randomization (ISR) technique is proposed to address code-injection attacks

- Key idea: an encoded version of software instructions is loaded into the memory and will be decoded by a key before being executed.

ISR: Instruction set mapping

```
31 ^ 12 => 23        6e ^ 82 => ec
c0 ^ ac => 6c        89 ^ ac => 25
50 ^ 7d => 2d        e3 ^ 03 => e0
68 ^ 9c => f4        50 ^ bc => ec
2f ^ a2 => 8d        53 ^ 90 => c3
2f ^ 55 => 7a        89 ^ ac => 25
73 ^ 38 => 4b        e1 ^ 7d => 9c
68 ^ cc => a4        99 ^ 97 => 0e
68 ^ 31 => 59        b0 ^ a2 => 12
2f ^ 0c => 23        0b ^ 0c => 07
62 ^ 7d => 1f        cd ^ 90 => 5d
69 ^ 91 => f8        80 ^ dc => 5c
```

ISR: Actual vs. intended code

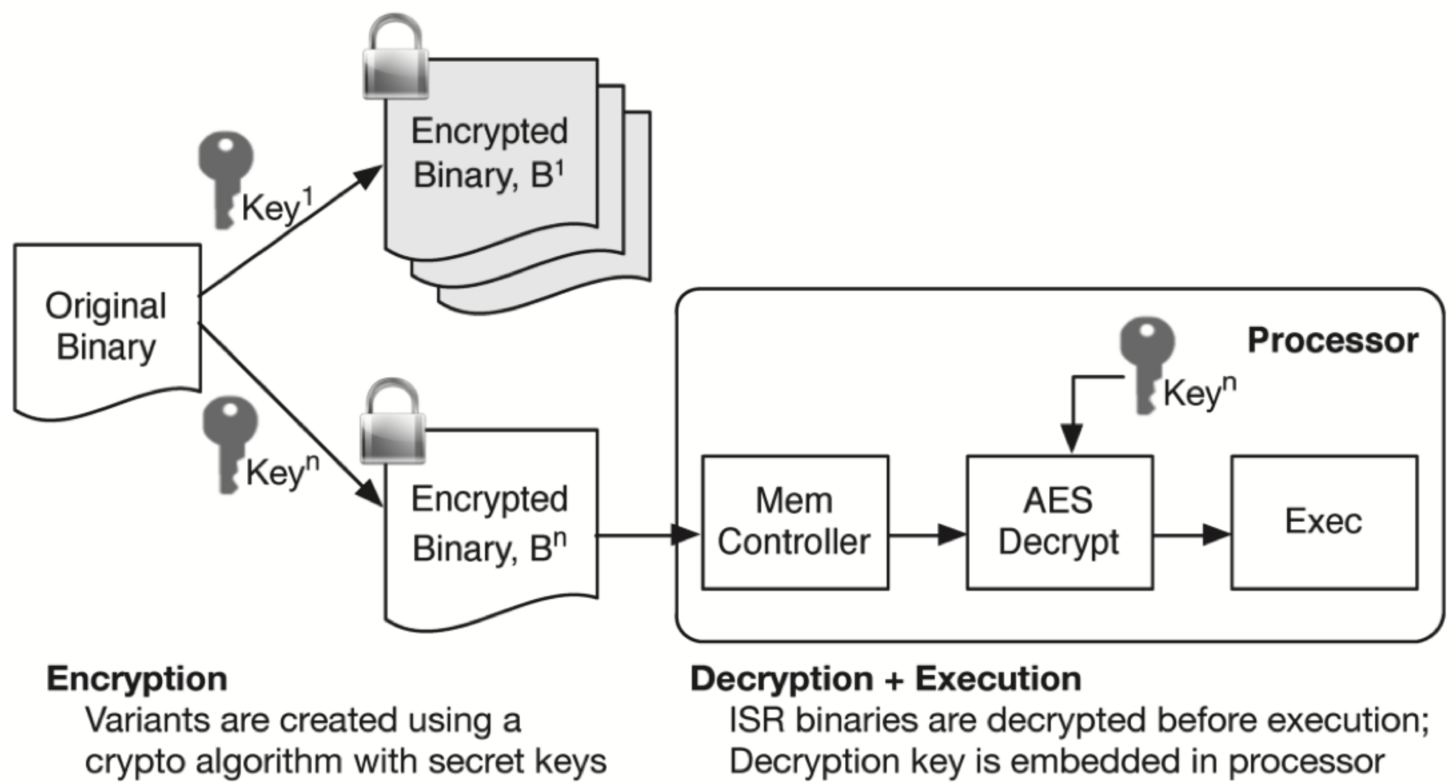
Code Actually Executed

23 6c 2d f4	and	0xfffffff4(%ebp,%ebp,1),%ebp
8d 7a 4b	lea	0x4b(%edx),%edi
a4	movsb	%ds:(%esi),%es:(%edi)
59	pop	%ecx
23 1f	and	(%edi),%ebx
f8	clc	
ec	in	(%dx),%al
25 e0 ec c3 25	and	$0x25c3ece0,%eax
9c	pushf	
0e	push	%cs
12 07	adc	(%edi),%al
5d	pop	%ebp
5c	pop	%esp
00 00	add	%al,(%eax)

Code Intended to Be Executed

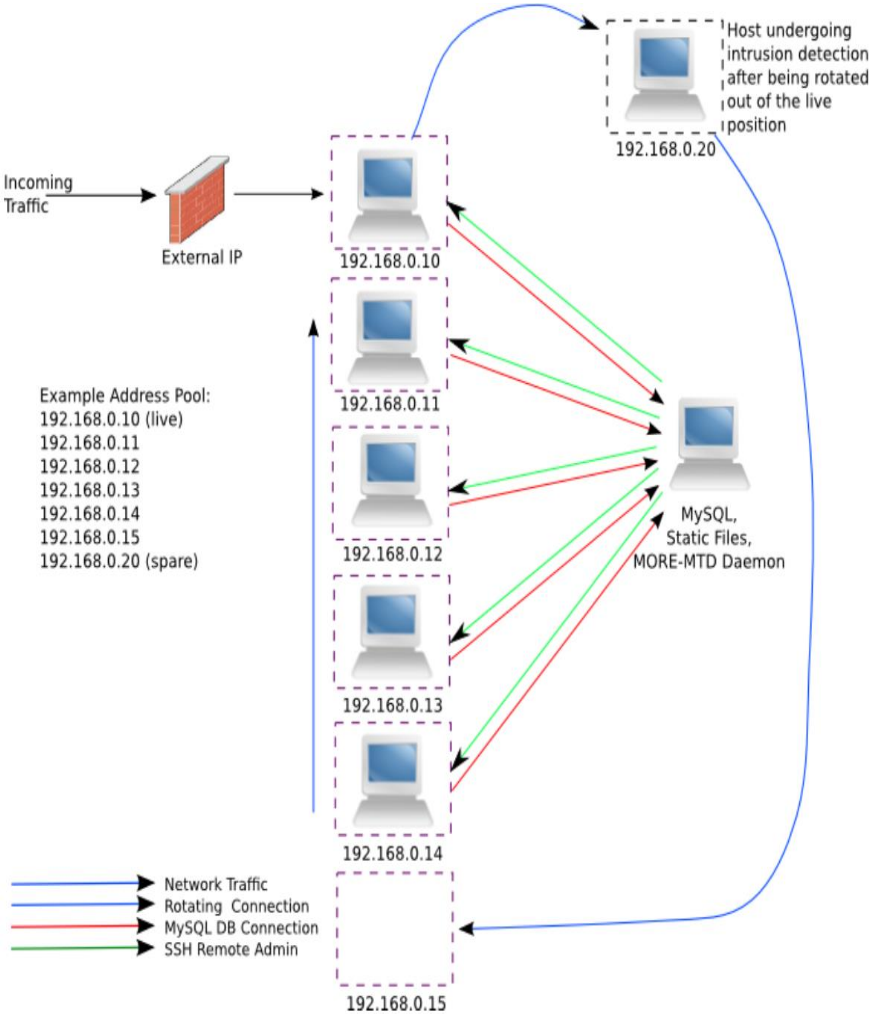
31 c0	xor	%eax,%eax
50	push	%eax
68 2f 2f 73 68	push	$0x68732f2f
68 2f 62 69 6e	push	$0x6e69622f
89 e3	mov	%esp,%ebx
50	push	%eax
53	push	%ebx
89 e1	mov	%esp,%ecx
99	cltd	
b0 0b	mov	$0xb,%al
cd 80	int	$0x80

ISR: Code encryption and decryption



Multiple rotational OS

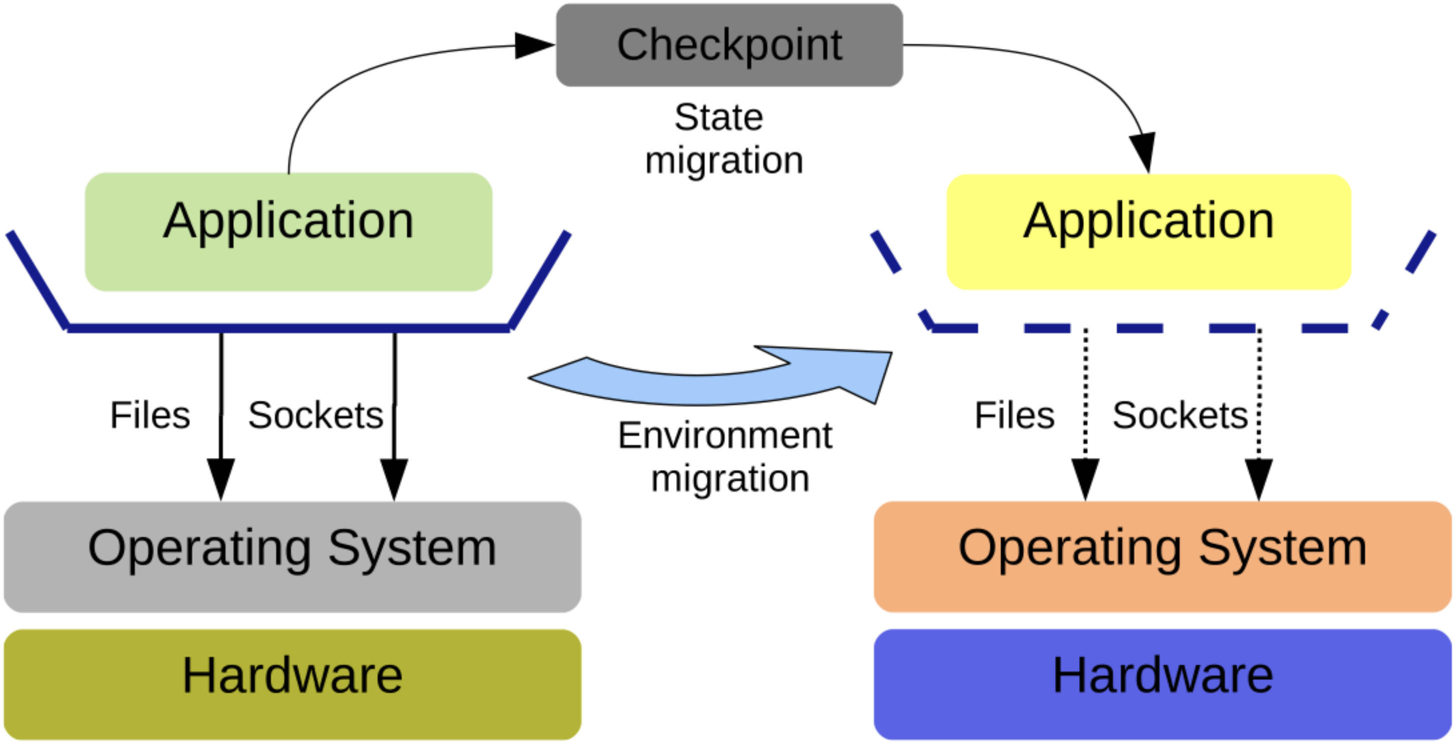
- Multiple VM hosts store shared data in a database and at one time only one of them will be mapped to an external IP address.
- The periodic rotation of VM hosts is controlled from an administrator machine running a daemon process, and the VM host that was previously in use is analyzed for evidence of intrusion and will be removed from rotation if compromised.



TALENT: Design goals

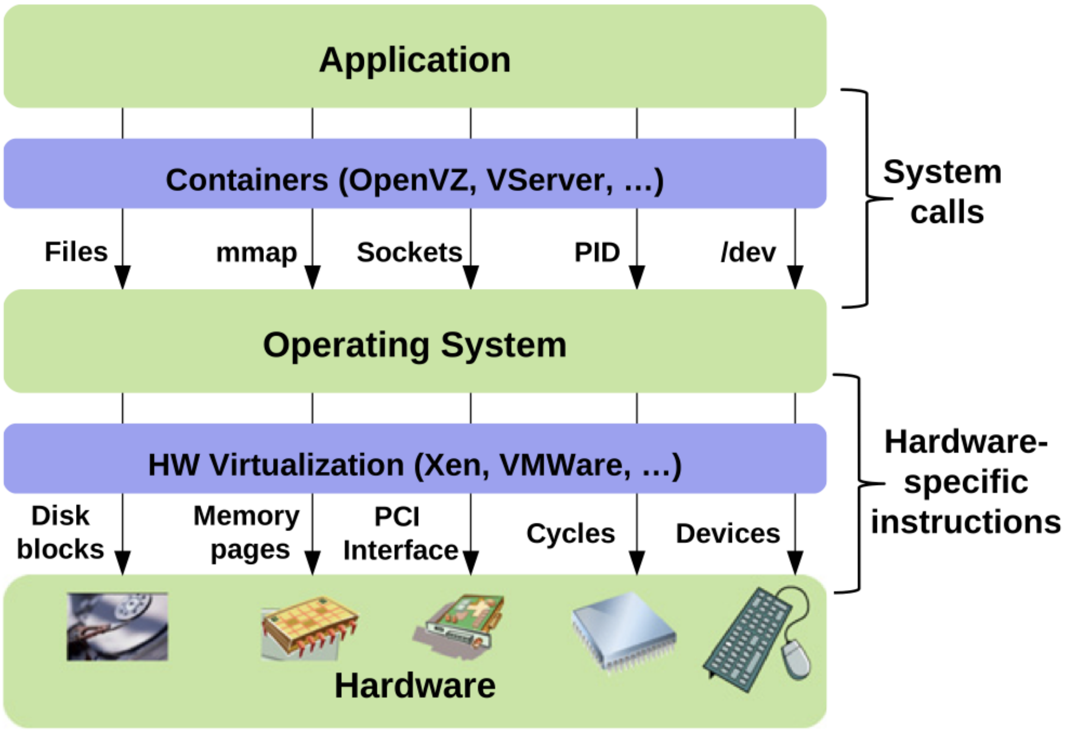
- **Heterogeneity at the instruction set architecture level, meaning that applications run on processors with different instruction sets.**
- **Heterogeneity at the OS level.**
- **Preservation of the state of the application, including the execution state, open files and sockets.**
- **Working with a general-purpose systems programming language such as C.**

TALENT: Migration



TALENT: OS-level virtualization

- In operating-system-level virtualization, the kernel allows for multiple isolated user-level instances, each of which is called a **container** (or jail or virtual environment).



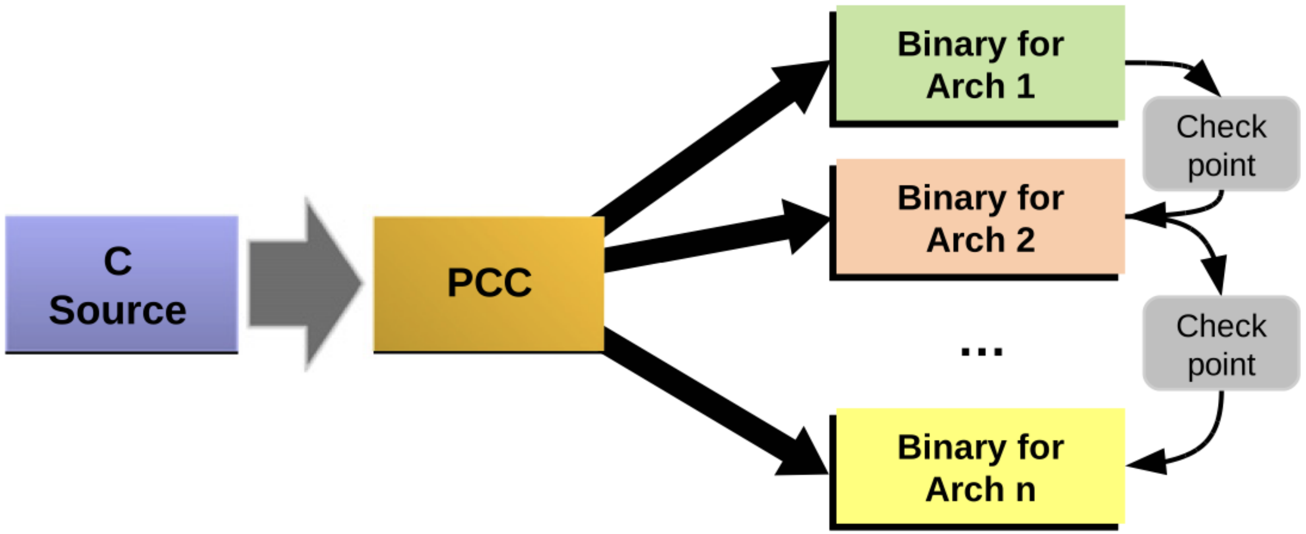
TALENT: Environment migration

- When migration is requested (as a result of a malicious activity or a periodic migration), TALENT migrates the container of the application from the source machine to the destination machine.
  - Done by synchronizing the filesystems of the destination container with the source container.

- To preserve **network connections** during migration, the IP address of the virtual network interface of the container is migrated to the new container. Then, the state of each TCP socket (sk_buff of the kernel) is transferred to the destination.
  - The network migration is seamless to the application, and the application can continue sending and receiving packets on its sockets.

TALENT: Running state migration

- The state of running programs must also be migrated.
- To do this, a method for checkpointing a running application must be implemented.
- After all the checkpointed program states are saved in checkpoint files, the state is migrated by simply mirroring the filesystem.



Dynamic Software

Software randomization

- There is a wide range of attacks exploiting software vulnerabilities, which requires precise understanding of the target software.
- By randomizing the implementation, software diversity introduces uncertainty in the target, increases the cost to attackers, and may provide an effective counter to side-channel attacks.
- **Examples**
  - ChameleonSoft
  - Marlin
  - GenProg

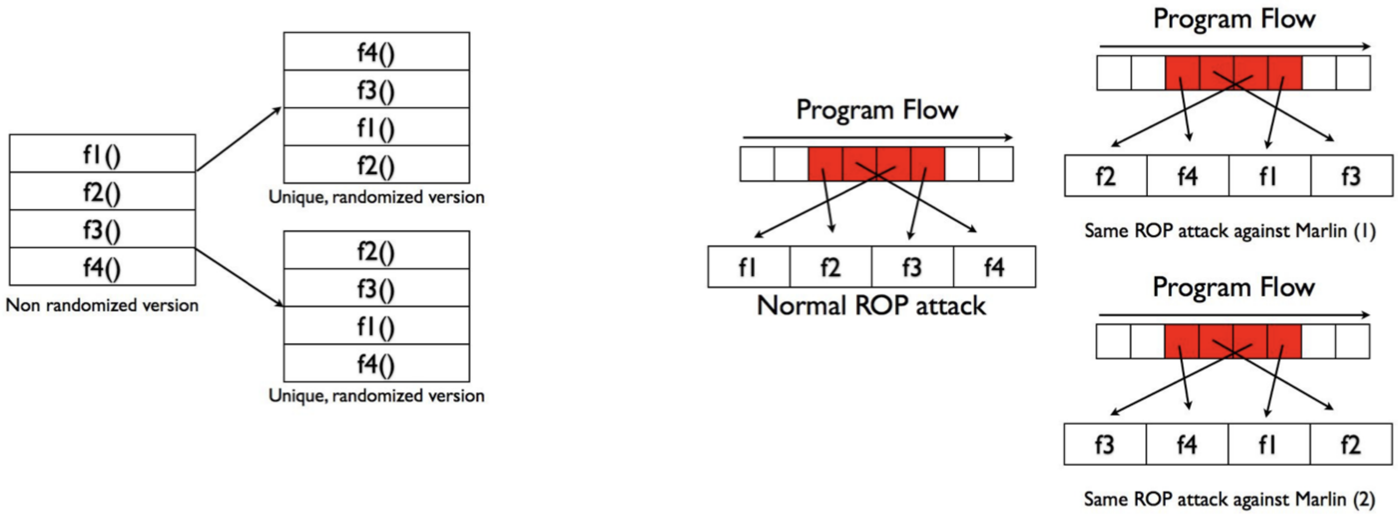
ChameleonSoft

- **ChameleonSoft** divides a complex software program into smaller tasks, each of which has a set of executable variants that are functionally equivalent but with different quality attributes (e.g., performance, robustness, and mobility).

- The executable variants can then be shuffled to change the attack surface in accordance with different security situations.

Marlin

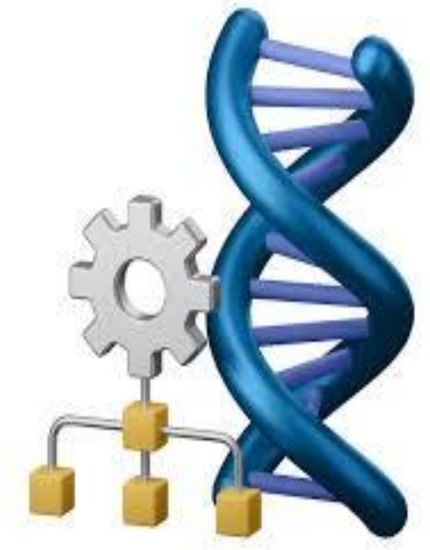
- Marlin breaks a software binary into function blocks and randomly shuffles the order. Such a process can be performed transparently at load time, which ensures every execution instance of the software to be unique.
- Markin can be used to defend against code reuse attacks, such as return-oriented programming (ROP).



(a) Randomization technique (b) Failure of ROP attacks

GenProg: Automatic software repair

- By utilizing an extended form of genetic programming (GP), GenProg is able to evolve a software program with identified vulnerabilities to a functionally equivalent variant that are no longer susceptible to the previous risks.

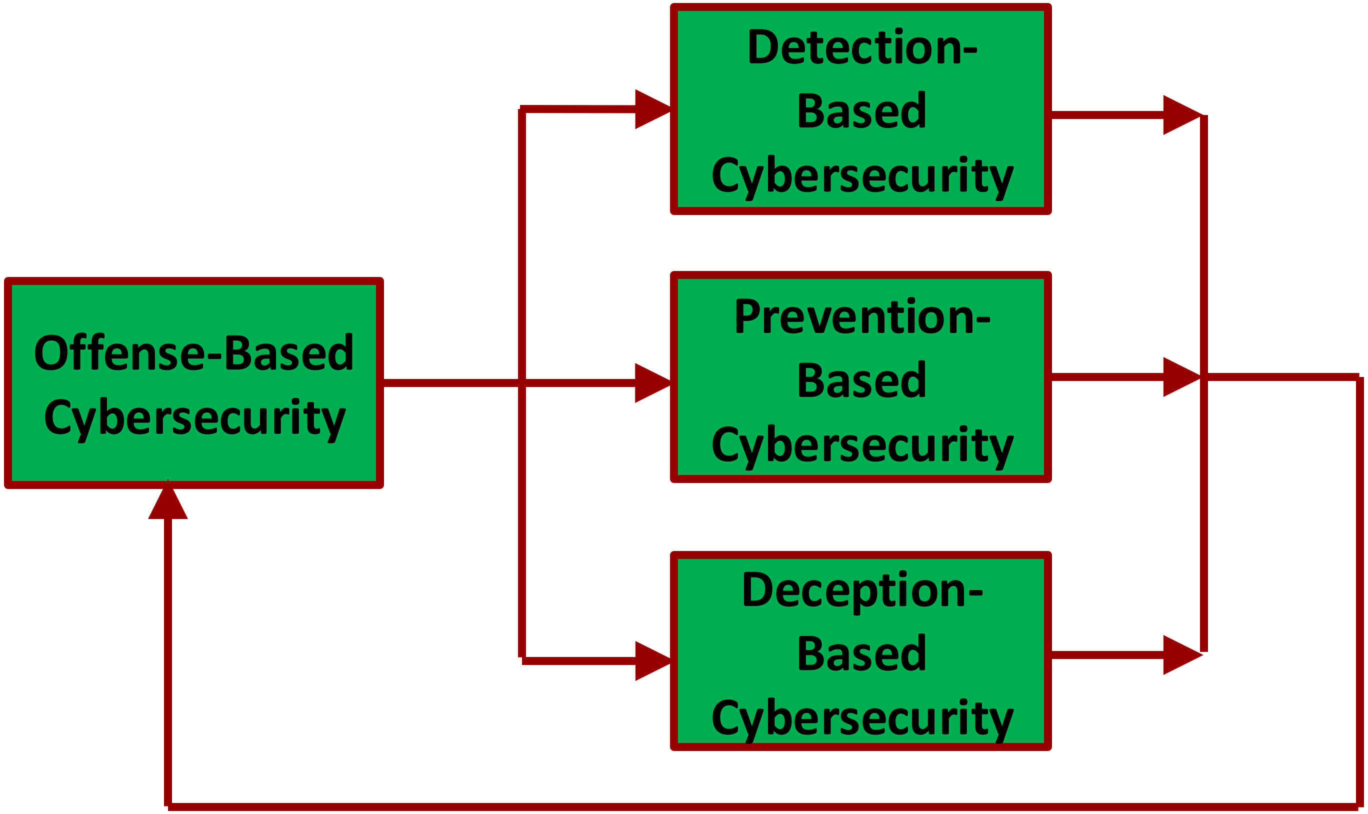


- GP uses computational analogs of biological mutation and crossover to generate new program variations, which are called variants. A user-defined fitness function evaluates each variant.

Summary of deception-based cybersecurity

- Art of cyber deception
- Honeypot
- ~~Honeytoken~~
- Moving target defense

Course structure



End of Lecture 20