BalCCon2k19 – Unchartered Waters

SANS

A  and a  celebrating 

Bojan Zdrnja | bojan.zdrnja@infigo.hr | @bojanz on Twitter

About me

- Bojan Ždrnja, Twitter: @bojanz
- CTO at INFIGO IS
  - https://www.infigo.hr
  - Penetration testing team lead
- SANS SEC542 instructor
  - Web application penetration testing
- SANS Internet Storm Center handler
  - https://isc.sans.edu

Why this presentation?

- I actually got tired of seeing all sorts of different risk ratings for SSL/TLS related vulnerabilities
- Something like this:

TLS/SSL service supports 64-bit block ciphers vulnerable to SWEET32 attack (CVE-2016-2183, CVE-2016-6329)

Vulnerability ID:	APP-02
Vulnerability type:	Insufficient Transport Layer Protection
Likelihood:	H
Impact:	H
Security risk:	HIGH

Why this presentation?

- In order to be able to assess the risk, we must know how these vulnerabilities work
- We will analyze the following most commonly reported SSL/TLS vulnerabilities
  - POODLE & BEAST
  - CRIME
  - RC4
  - SWEET32
- … and see how viable their exploitation is

SSL/TLS introduction

- The main goal of SSL/TLS is to enable private communication over insecure media
- SSL/TLS sessions are secured with a number of algorithms
  - Key exchange and authentication algorithms
  - Message encryption
    - DES/3DES/AES or RC4/ChaCha
  - Message authentication

Block ciphers

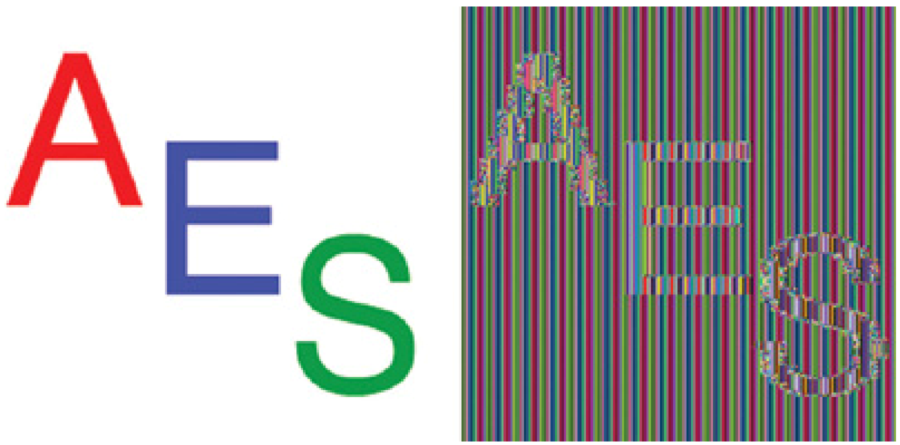
- DES and AES are the most common examples of block ciphers
  - Everything encrypted must be divided into blocks
    - Blocks are typically 8 bytes (DES/3DES) or 16 bytes long (AES)
  - There are different block cipher modes
    - ECB (Electronic Code Book)
    - CBC (Cipher Block Chaining)
    - CTR (Counter)
    - GCM (Galois/Counter Mode)
    - ...

Block ciphers

- DES and AES are the most common examples of block ciphers
  - Everything encrypted must be divided into blocks
    - Blocks are typically 8 bytes (DES/3DES) or 16 bytes long (AES)
  - There are different block cipher modes
    - ECB (Electronic Code Book)
    - CBC (Cipher Block Chaining)
    - CTR (Counter)
    - GCM (Galois/Counter Mode)
    - …

Block ciphers

- ECB is bad, we already know that



- Every block, encrypted with a same key always gives the same output

The SUPER fantastic XOR

$$A \oplus B = B \oplus A$$

$$A \oplus B = C$$

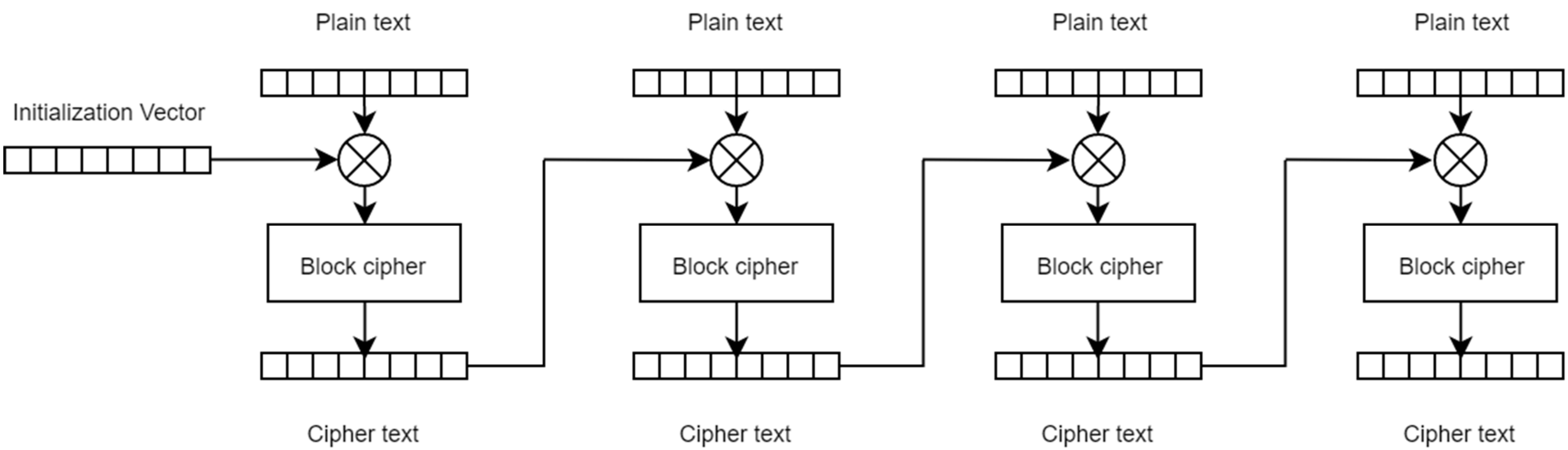
$$A = B \oplus C$$

$$A \oplus 0 = A$$

$$A \oplus A = 0$$

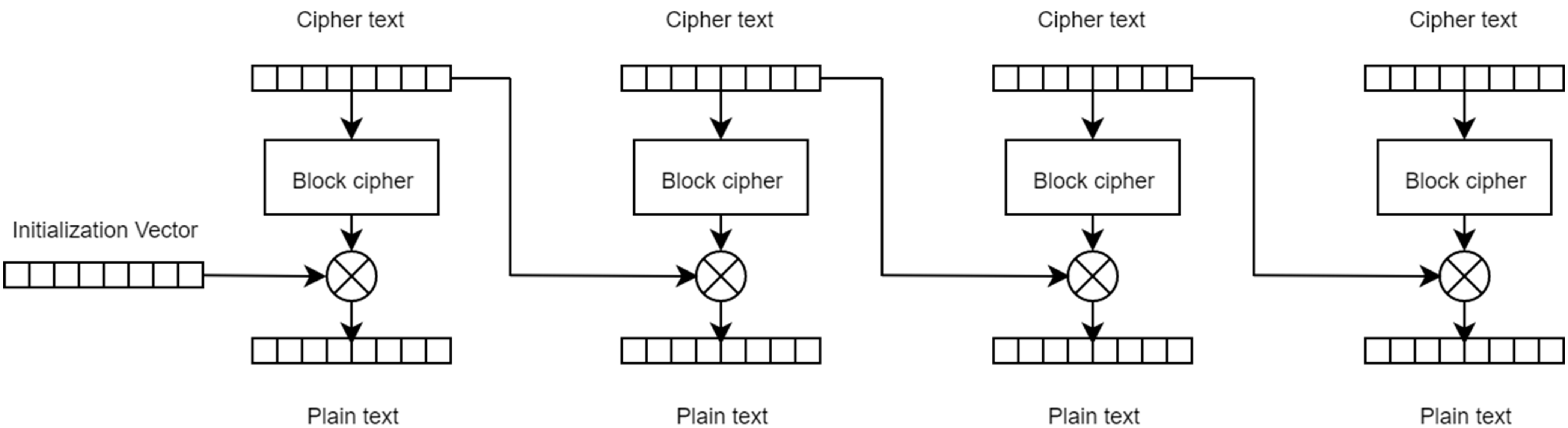
Block ciphers (CBC)

- CBC was the doom of SSLv3 (and TLSv1.0), let's see why
- Encryption in CBC:



Block ciphers (CBC)

- CBC was the doom of SSLv3 (and TLSv1.0), let's see why
- Decryption in CBC:



Block ciphers

- Let’s analyze what a simple request looks like when handled by SSLv3
  - A GET / request for isc.sans.edu, with a PHPSESSID cookie
  - Encrypted with 3DES

G	E	T		/		H	T
T	P	/	1	.	1	\r	\n
H	o	s	t	:		i	s
c	.	s	a	n	s	.	e
d	u	\r	\n	C	o	o	k
i	e	:		P	H	P	S
E	S	S	I	D	=	3	a
3	k	o	w	1	1	4	a
\r	\n	\r	\n				

Block ciphers

- First critical issue: SSLv3 computes MAC and then encrypts the message
  - SHA1 is 160 bits = 20 bytes

G	E	T		/		H	T
T	P	/	1	.	1	\r	\n
H	o	s	t	:		i	s
c	.	s	a	n	s	.	e
d	u	\r	\n	C	o	o	k
i	e	:		P	H	P	S
E	S	S	I	D	=	3	a
3	k	o	w	1	1	4	a
\r	\n	\r	\n	M	M	M	M
M	M	M	M	M	M	M	M
M	M	M	M	M	M	M	M

Block ciphers

- Now, we need padding
  - The message must always be in multiple number of blocks and must have padding

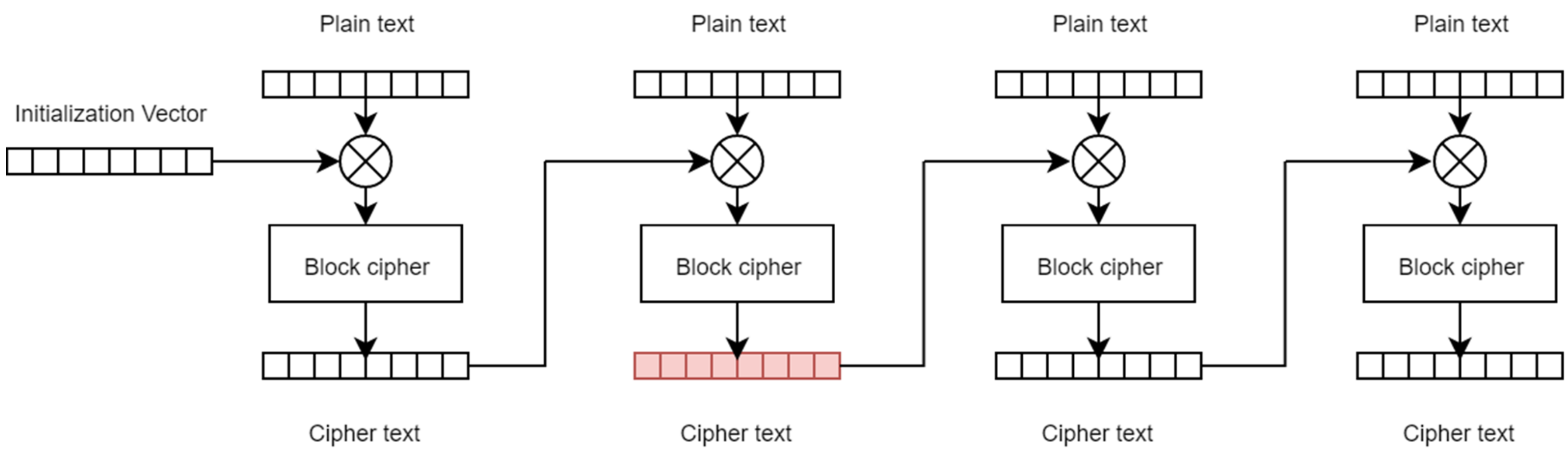
G	E	T		/		H	T
T	P	/	1	.	1	\r	\n
H	o	s	t	:		i	s
c	.	s	a	n	s	.	e
d	u	\r	\n	C	o	o	k
i	e	:		P	H	P	S
E	S	S	I	D	=	3	a
3	k	o	w	1	1	4	a
\r	\n	\r	\n	M	M	M	M
M	M	M	M	M	M	M	M
M	M	M	M	M	M	M	M
							0x7

Block ciphers

- Attack idea
  - When the last block is padding, the very last byte must be 0x07 (for AES, it must be 0x0f)
  - Bytes before padding in the last block are garbage and are ignored (in SSLv3!)
  - If the last block decrypts to anything else, MAC will be incorrect
    - Remember, MAC is always appended to a message, and after MAC padding is added
  - Crucial issue:
    - When the MAC is wrong, this will be signaled by SSLv3
    - This allows us to use the server as an Oracle!

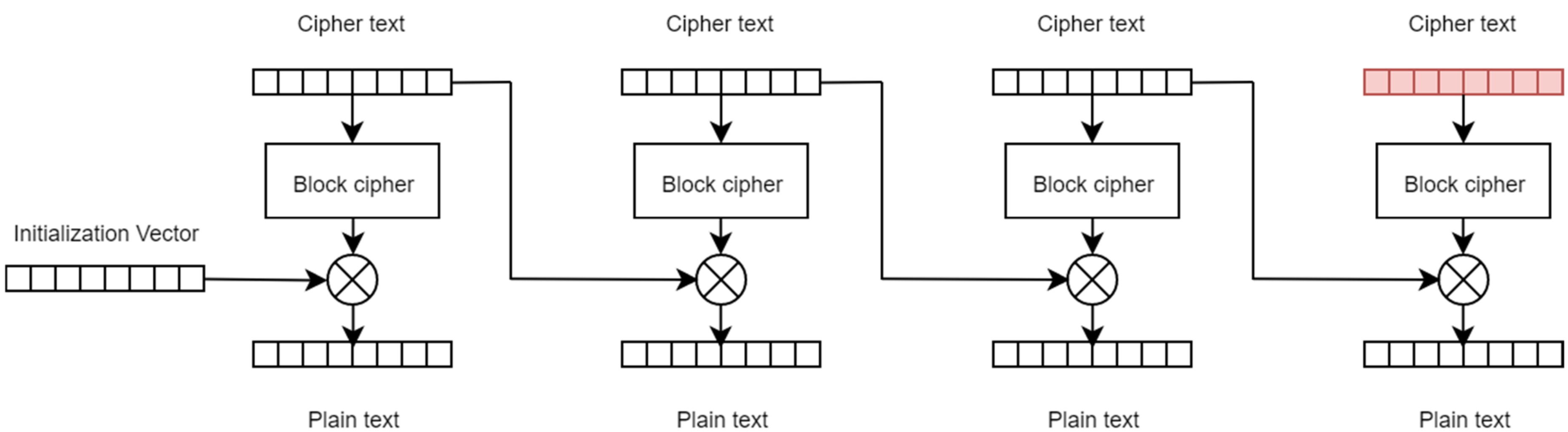
Block ciphers

- The attacker must first run a Man-in-the-Middle attack
- Then we carefully copy a block



Block ciphers

- The attacker must first run a Man-in-the-Middle attack
- Then we carefully copy a block



Block ciphers

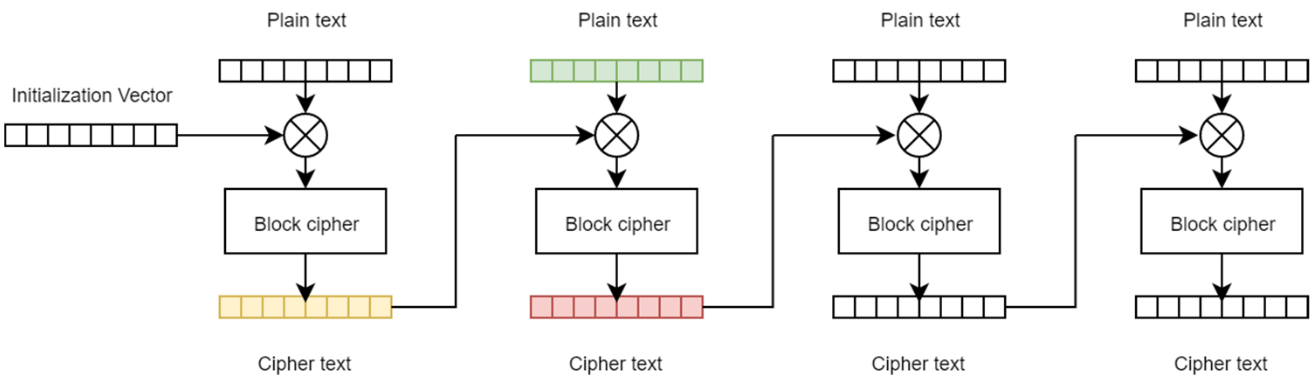
- What happens now?
  - The server first decrypts everything
    - If the last byte decrypted to anything other than 0x07:
      - Everything fails, the MAC is wrong
      - We get a TLS alert!
        - Alert (21) = Decryption failed

```
▸ Frame 1948: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0
▸ Ethernet II, Src: Vmware_94:0a:c5 (00:0c:29:94:0a:c5), Dst: Vmware_d7:2e:f1 (00:0c:29:d7:2e:f1)
▸ Internet Protocol Version 4, Src: 192.168.210.135, Dst: 192.168.210.136
▸ Transmission Control Protocol, Src Port: 52806, Dst Port: 4443, Seq: 1039, Ack: 1254, Len: 29
▾ Secure Sockets Layer
  ▾ SSLv3 Record Layer: Encrypted Alert
      Content Type: Alert (21)
      Version: SSL 3.0 (0x0300)
      Length: 24
      Alert Message: Encrypted Alert
```

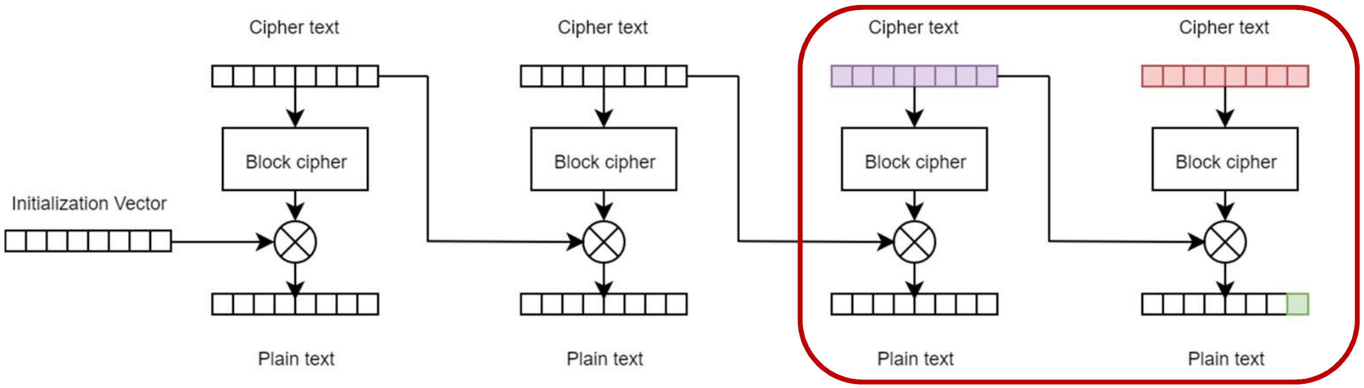
Block ciphers

- What happens now?
  - What if the attacker gets lucky and the last byte is 0x07?
  - Decryption will be successful
    - Remember, everything else in the last block is ignored
    - Let us visualize this

Block ciphers



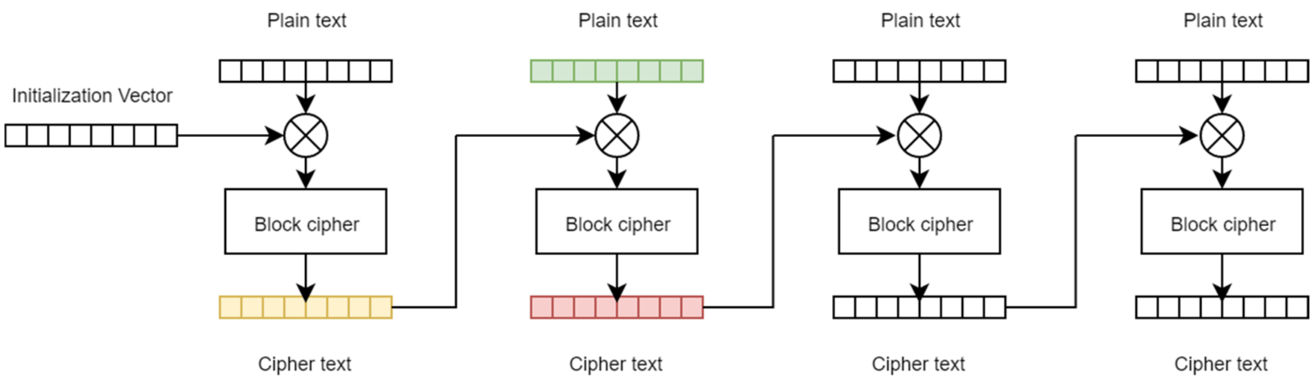
$$P_n = D_k(C_n) \oplus C_{n-1}$$



i = the block we want to guess

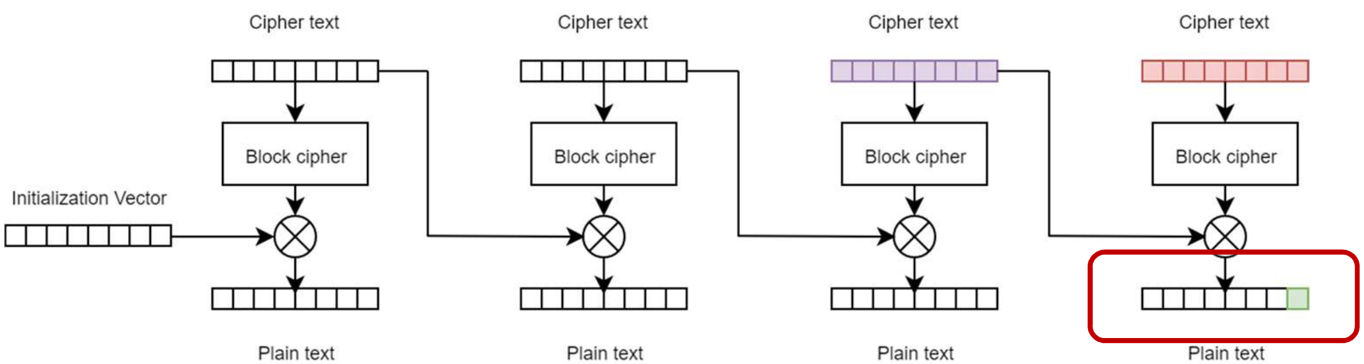
n = the last block

Block ciphers



$$P_n = D_k(C_n) \oplus C_{n-1}$$

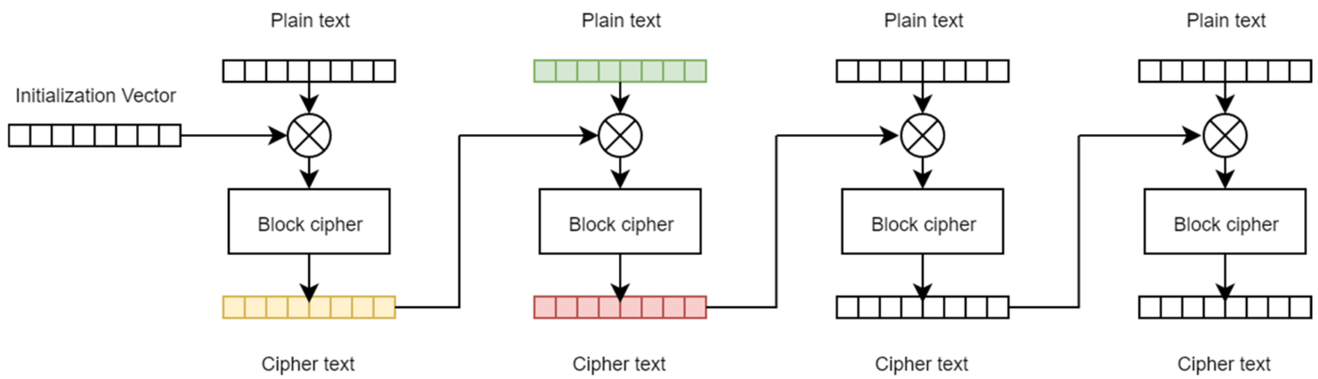
$$\boxed{xxxxxxx7 = D_k(C_n) \oplus C_{n-1}}$$



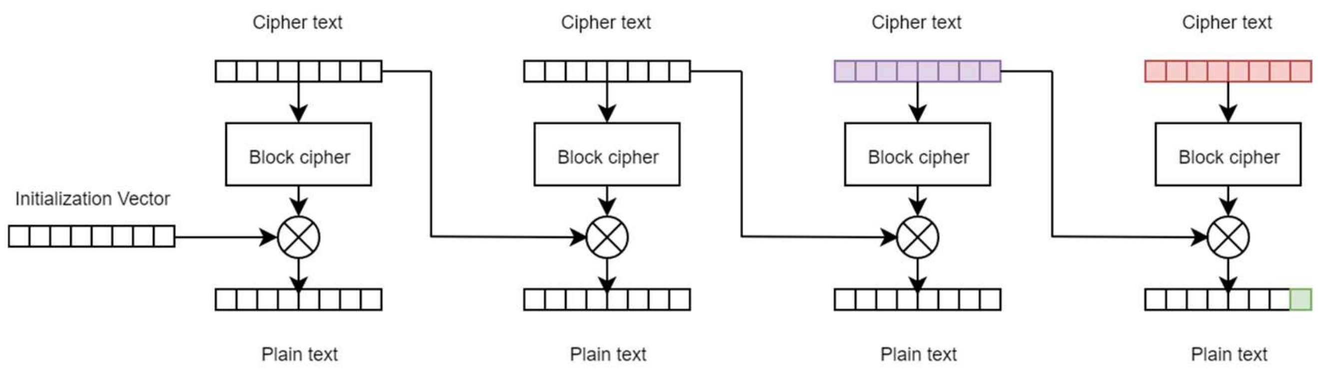
i = the block we want to guess

n = the last block

Block ciphers



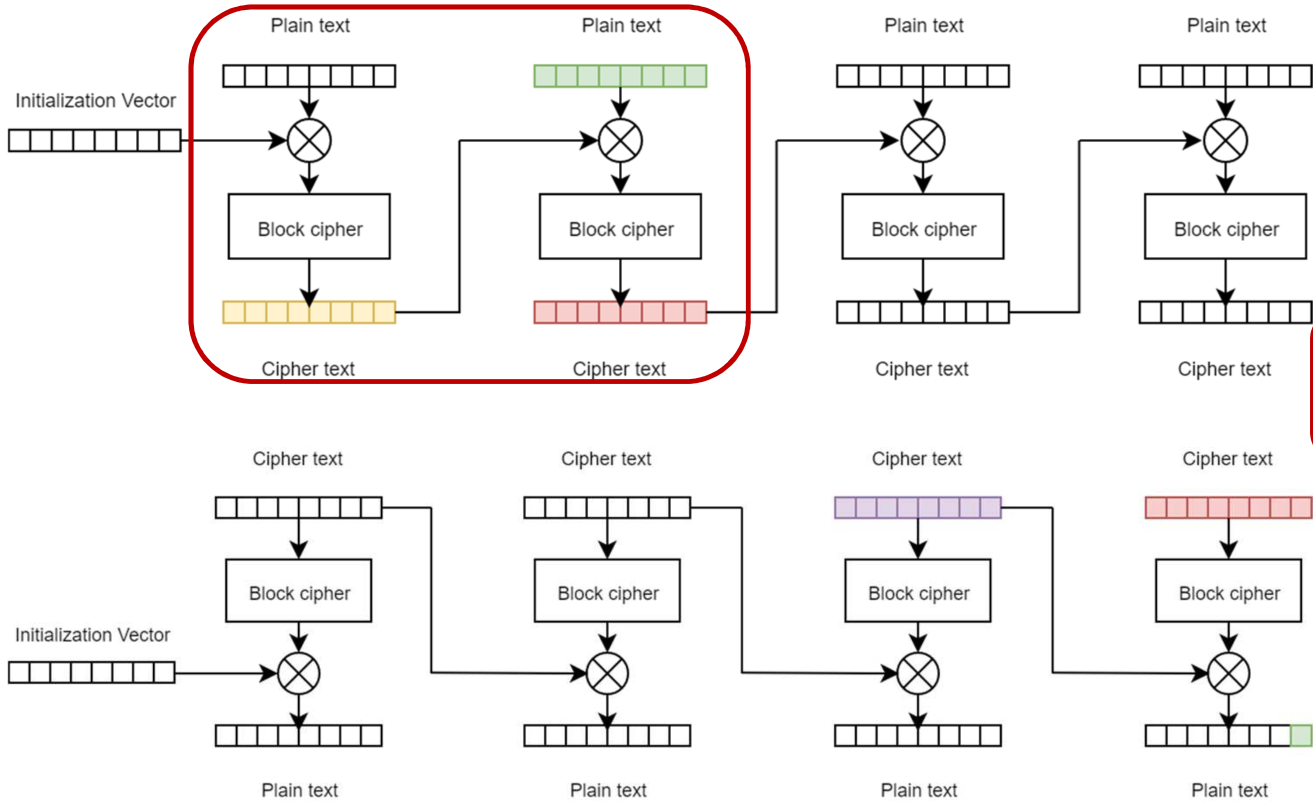
$$P_n = D_k(C_n) \oplus C_{n-1}$$
$$xxxxxxx7 = D_k(C_n) \oplus C_{n-1}$$
$$D_k(C_n) = xxxxxxx7 \oplus C_{n-1}$$



i = the block we want to guess

n = the last block

Block ciphers



$$P_n = D_k(C_n) \oplus C_{n-1}$$

$$xxxxxxx7 = D_k(C_n) \oplus C_{n-1}$$

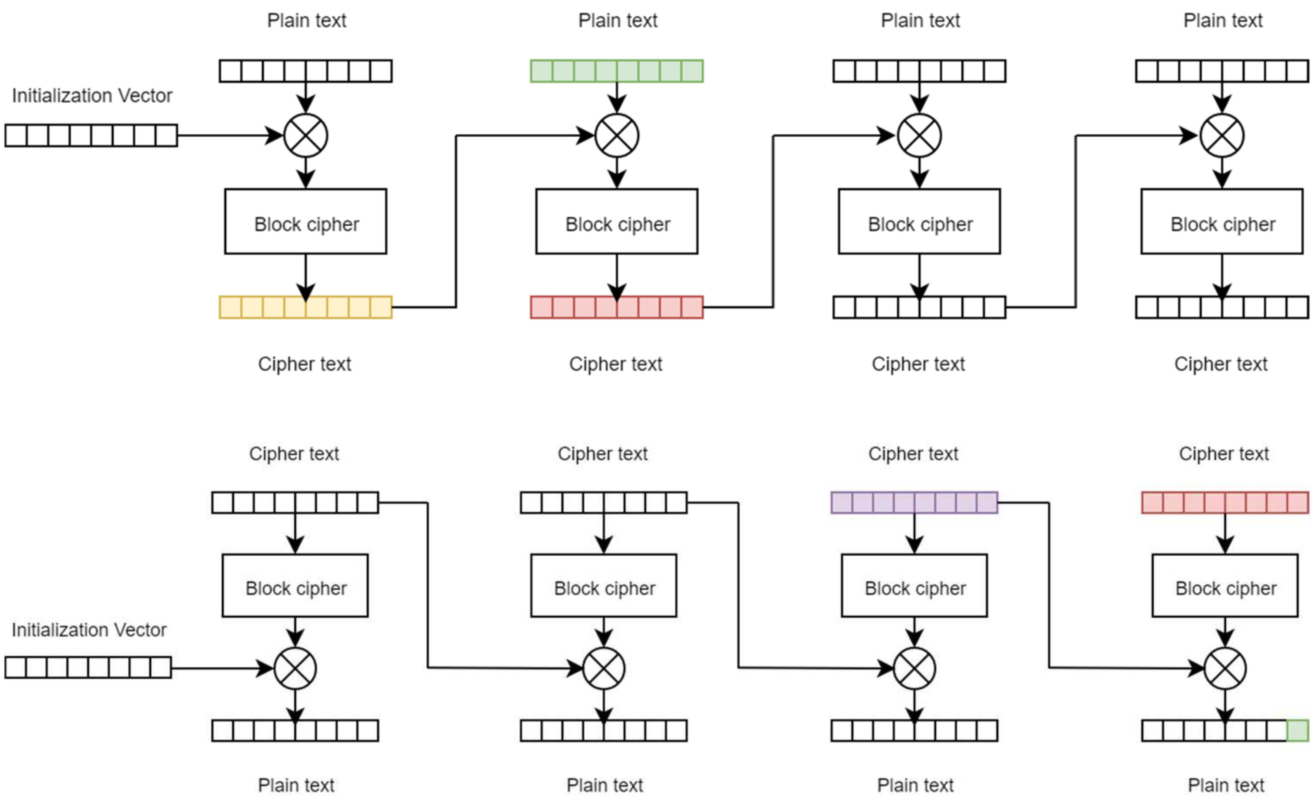
$$D_k(C_n) = xxxxxxx7 \oplus C_{n-1}$$

$$P_i \oplus C_{i-1} = xxxxxxx7 \oplus C_{n-1}$$

i = the block we want to guess

n = the last block

Block ciphers



$$P_n = D_k(C_n) \oplus C_{n-1}$$
$$xxxxxxx7 = D_k(C_n) \oplus C_{n-1}$$
$$D_k(C_n) = xxxxxxx7 \oplus C_{n-1}$$
$$P_i \oplus C_{i-1} = xxxxxxx7 \oplus C_{n-1}$$
$$P_i = C_{i-1} \oplus xxxxxxx7 \oplus C_{n-1}$$

i = the block we want to guess

n = the last block

POODLE demo

- This is the POODLE vulnerability
  - Padding Oracle On Downgraded Legacy Encryption
- Attack requirements
  - The attacker must run a Man-in-the-Middle
  - The attacker must be able to run arbitrary JavaScript code in the victim's browser
    - Typically done by injecting JavaScript in HTTP content
  - The attacker now must observe errors from the server that is running SSLv3 (SSL alerts)

POODLE demo

- What will we send?

```
POST /aaa HTTP/1.1
Host: isc.sans.edu
Cookie: PHPSESSID=1234567890

bbbbbb
```

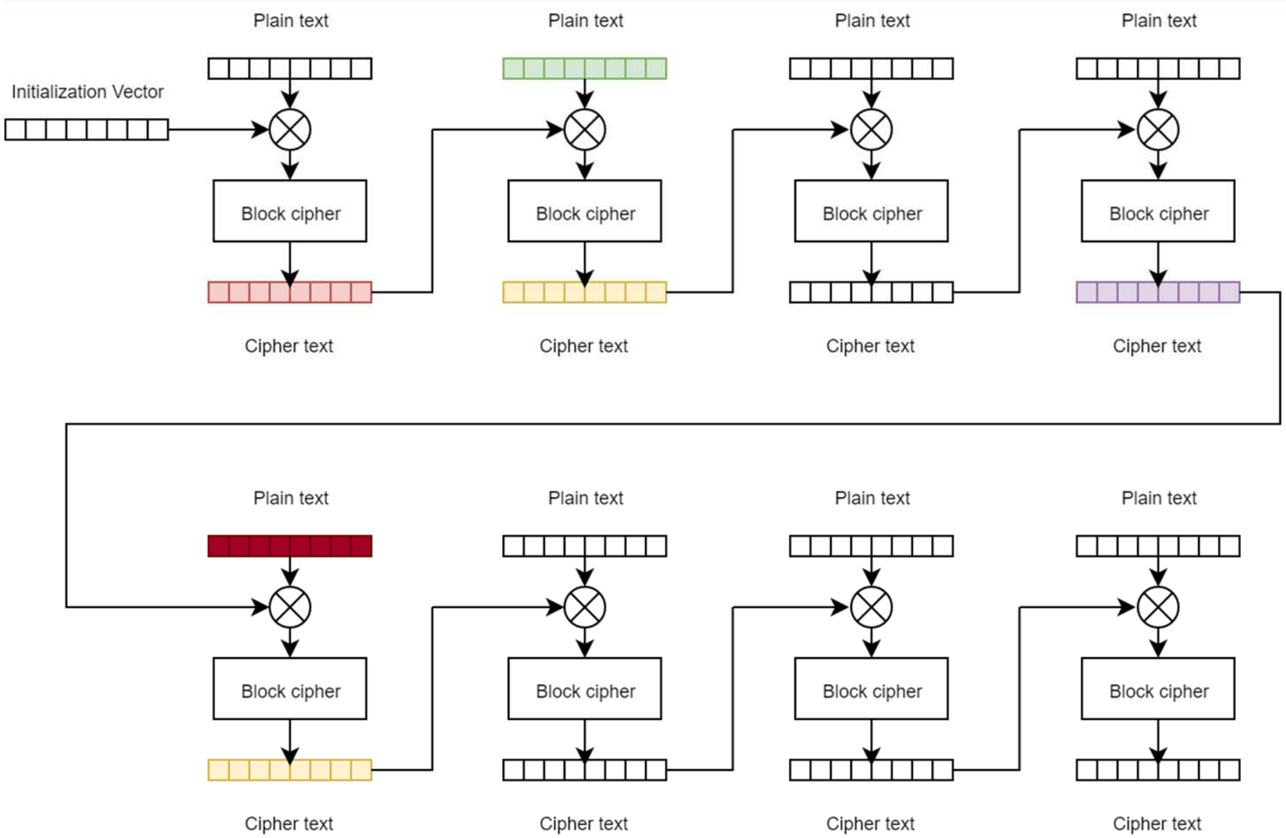
POODLE demo

- Based on https://github.com/mpgn
  - Twitter https://twitter.com/mpgn_x64
  - Modified to work for the demo
    - Demo files available on my github, https://github.com/bojanisc
    - Ubuntu 14.04LTS will be used for the demo
    - Mozilla Firefox 30, with SSLv3 support enabled
  - In the demo the attacker is positioned as a proxy
    - Simulates a successful Man-in-the-Middle attack

BEAST

- Browser Exploit Against SSL/TLS
- BEAST was published before POODLE
- Based on the same principles of XOR-ing plain text and encrypted blocks in CBC algorithms
  - This time the browser is the Oracle
  - The attacker will try to guess a block
    - That's impossible really, so let's settle down for guessing a single byte
    - Remember our 8-byte blocks?
      - SESSID=0, SESSID=1, SESSID=2
      - Now we need to guess only 1 byte at a time – this is doable

BEAST



$$C_i = E_k(P_i \oplus C_{i-1})$$

$$\mathbf{P_2} = G_i \oplus C_{i-1}$$

$$C_i = E_k(\mathbf{P_2} \oplus C_n)$$

$$C_i = E_k(G_i \oplus C_{i-1} \oplus C_n)$$

$$P_i \oplus C_{i-1} = G_i \oplus C_{i-1} \oplus C_n$$

$$G_i = P_i \oplus C_n$$

CRIME

- CRIME (Compression Ratio Info-Leak Made Easy) is an attack against SSL compression and SPDY
- Works by exploit the leak that happens when compressing data
  - While compression functions can be sophisticated, the basics is that the length depends on the content that is compressed
    - Compressed strings "aaaa" and "aaab" will have different length
    - An attacker can exploit this by guessing content

CRIME

- What can we guess – cookies of course
- Attack requirements
  - The attacker must run a Man-in-the-Middle
  - The attacker must be able to run arbitrary JavaScript code in the victim's browser
    - Typically done by injecting JavaScript in HTTP content
  - The attacker now monitors length of sent data
- Demo with zlib()

RC4

- With RC4 the issues is actually in the encryption itself
  - Not related to the version of SSL or TLS – all are equally bad
- Research published in 2015 by Mathy Vanhoef and Frank Piessens
- They found that there are biases in RC4 encryption
  - When a lot of data encrypted with RC4 is analyzed, biases can be detected
    - I.e. in consecutive values (0,0) and (0,1)
    - We play a game of guessing

RC4

- Can be especially powerful against HTTPS requests
  - Remember we have a lot of surrounding content that is known to the attacker
    - HTTP request line, browser headers etc
  - By observing network traffic, the attacker can try to probabilistically predict plaintext
  - Attack needs to satisfy the following again:
    - The attacker needs to launch a Man-in-the-Middle attack against a victim
    - The attacker must be able to cause the victim's browser to issue (a lot) of HTTP requests
      - It will be JavaScript again

RC4

- Results from 2015 maybe do not look too scary
- In order to decrypt a 16-character cookie they needed to perform the following:
  - Send ~6 * 2^{27} requests!
  - This amounts to ~300 GB of traffic
  - It took them 52 hours to generate this traffic with speed of about 4450 requests / second
- Remember one thing with crypto attacks: they only get better with time!

SWEET32

- SWEET32 is again a crypto issue
- The idea is based on collision attacks in CBC algorithms
  - See a pattern here?
- If two blocks have the same output (collision) we can reveal the XOR of two plaintext blocks

SWEET32

- Birthday paradox
  - In a room of 23 people, there is a 50% chance that two of them share the same birthday
- CBC leaks plaintext after $2^{n/2}$ blocks encrypted with the same key
  - So we must just rekey frequently right?
  - Unfortunately many TLS libraries or old browsers do not do that
- With a 64-bit cipher, first collision around 32GB

SWEET32

- SSL3, TLS1.0 do not rekey
- TLS1.1, TLS1.2 rekey after 2^{78} requests
- So how bad this is?
  - Remember with HTTP that all requests are very similar
    - Sensitive data (cookie) is almost always at the same position
  - Attack needs to satisfy the following again:
    - The attacker needs to launch a Man-in-the-Middle attack against a victim
    - The attacker must be able to cause the victim's browser to issue (a lot) of HTTP requests
      - It will be JavaScript again

SWEET32

- Additionally, the target server must support very long sessions
  - HTTP/1.1 Keep-Alive allows reusing a connection
  - Defaults for servers should be around 200 requests, but many servers allow long sessions
- Practical attacks from 2016:
  - Send ~300 GB in about 30.5 hours
- Remember one thing with crypto attacks: they only get better with time!

How to test

- Several great and stable tools
- Nmap with its NSE scripts
- The testssl.sh utility
- Qualys SSL labs for public web sites
  - Amazing amount of information
  - Make sure you select the "Do not show" button so your scan is not listed on the front page
  - Keep in mind that Qualys will collect this data and needs to be able to connect to your web site

nmap

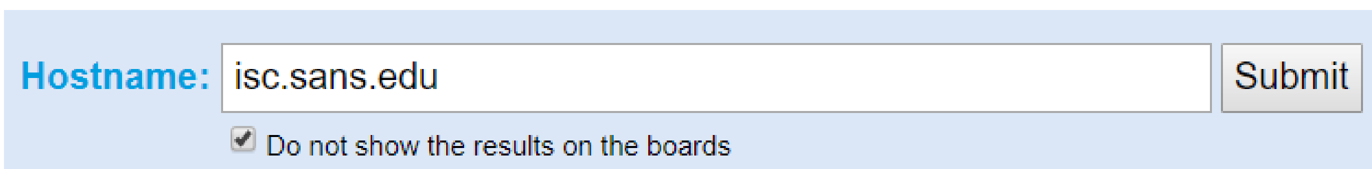
- Nmap comes with the fantastic ssl-enum-ciphers script
- Use it on all sites to identify supported ciphers
- Some tips&tricks
  - If it is a non-standard service, prepend the + character to ensure that the script runs
  - The script will not check for SSLv2 – another script (sslv2) must be run for that
  - Scores are based on both key exchange and encryption algorithms
  - Always make sure you run the very latest version of nmap

testssl.sh

- Great free command line tool that can check for virtually every SSL/TLS vulnerability
  - Supports STARTTLS for many protocols as well
  - Comes with its own build of openssl to ensure that all ciphers that are required are supported
  - No need to install, simply drop and use
  - New features added constantly

Qualys SSL labs

- https://www.ssllabs.com/ssltest/
- Nice to use if you are testing a publicly available web site
- Not sure if I want to share this data with Qualys though
- Click “Do not show the results on the boards” if you use it



Qualys SSL labs

You are here: Home > Projects > SSL Server Test > isc.sans.edu

SSL Report: isc.sans.edu (204.51.94.153)

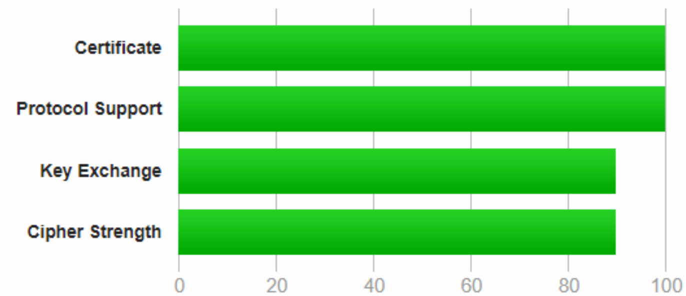
Assessed on: Mon, 17 Jun 2019 01:20:33 UTC | **HIDDEN** | Clear cache

Scan Another »

Summary

Overall Rating

A+



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

Conclusion

- Almost all vulnerabilities require an attacker to successfully launch a Man-in-the-Middle attack
- POODLE is a real threat, disable SSLv3
  - For browser based applications – the attacker needs to be able to make the victim issue arbitrary requests!
- CRIME and BEAST should be fixed by modern browsers
- RC4 is a real threat, disable it
- SWEET32 – we can probably live with it for now
- Keep in mind that crypto attacks only get better with time!

Questions?

https://isc.sans.edu