# Dynamic Cryptographic Backdoors Part II Taking Control over the TOR Network

Eric Filiol (speaker) - Oluwaseun Remi-Omosowon - Leonard Mutembei
filiol@esiea.fr,remi-omosowon@et.esiea-ouest.fr
http://sites.google.com/site/ericfiliol

ESIEA - Laval
Operational Cryptology and Virology Lab $(C+V)^O$

PacSec 2011 - Tokyo Japan

# Outline

# Cryptanalysis reality

- What does "to break cryptography" means?
- Use the "armoured door on a paper/cardboard wall" syndrom?
  - The environment (O.S, user, network architecture...) is THE significant dimension.
- Make sure that everyone uses the standards/norms/tools you want to impose (one standard to tie up them all).
- Standardization of mind and cryptographic designs/implementation.
- Think in a different way and far from the official cryptographic thought.
- To break a system means actually and quickly accessing the plaintext whatever may be the method.

# Dynamic Cryptographic Backdoors Part I Content

- Presented at CanSecWest 2011 (sequel of H2HC 2010 and Black Europe 2010).
- We have shown how to
  - Bypass IPSec-based encrypted networks (with or without Tempest hardening).
  - Break operationally unknown, weakly implemented stream ciphers or block ciphers in stream cipher mode.
  - Application to IP encryptors
- Let us now present how to use all of this to take control over the TOR network.
- Our working operational scenario:
  - a non-democratic country which wants to monitor all its political opponents (outside and inside the country).
  - any small/medium size group of bad guys.

# Summary of the talk

# Outline

# Introduction

- How to bypass security enforced in very secure encrypted protocols (e.g. IP encrypting routers with systematic padding)?
- **First solution**: exploit the fact that many encryption algorithms rely on the operating system primitives to generate secret keys (e.g. Microsoft cryptographic API).
- **Second solution**: modify the cryptographic algorithm on-the-fly in memory: its mode of operation and/or its mathematical design.
- The algorithm is not modified on the hard disk (no static forensics evidence).
- The trapdoor has a limited period of time and can be replayed more than once.
- In both cases, the encryption has been weakened in such a way that the attacker has just to intercept the ciphertext and perform the cryptanalysis.

# OS Level Dynamic Trapdoors

- Here we considered strong cryptosystems (AES, TrueCrypt, GPG/PGP...).
- However the security at the operating system/network level is not perfect.
- What is it possible to do with a simple malware in a more sophisticated, coordinated attack?
- The "static (mathematical) security" remains unquestioned!
- Just create dynamically periods of time during which the encryption system is weak.
- All techniques tested and validated in real environments/conditions.

# Program Interaction Control

- Here we exploit the fact that very often, the message key $K_m$ is built from data provided by external programs.
  - Message counter, message key, session key...
  - Initialization vectors for block ciphers.
  - Integer nonces.
- Most of the time the resources involved are in the Windows API.
  - They provide random data required by the encryption application to generate message keys and IVs
- You then just have to hook the API function involved.
- Same approach for other equivalent resources (network infrastructure, key infrastructure, network-based key management...).

# Hooking the CryptGenRandom function

- Drawn from a real case (see further).
- A malicious DLL is injected in some (suitable) processes. This DLL hooks the CryptGenRandom function (included in Microsoft's Cryptographic Application Programming Interface).

**CryptGenRandom function**

```
BOOL WINAPI CryptGenRandom(
__in HCRYPTPROV hProv,
__in DWORD dwLen,
__inout BYTE *pbBuffer
);
```

- A timing function checks whether we are in the time window given as parameter $sTime(12, 00, 14, 00)[...]$. will hook the CryptGenRandom function between noon and 2pm only.

# Hooking the `CryptGenRandom` function (2)

- The integer (random data) returned by `CryptGenRandom` is modified by the function `HookedCryptGenRandom`.
  - They provide random data required by the encryption application to generate message keys and IVs
- You then just have to hook the API function involved.
- Same approach for other equivalent resources (key infrastructure, network-based key management...).
- On Bob's side, the ciphertext can still be deciphered.
  - Most protocols send $K_m$ and IVs with the message.
- You can also make $K$ and IVs reset more or less frequently to have parallel chunks of ciphertexts.

# Hooking the CryptGenRandom function (3)

Generate fixed message key $0x1212121212121212$

**HookedCryptGenRandom function**

```
BOOL WINAPI HookedCryptGenRandom(HCRYPTPROV hProv, DWORD
dwLen, BYTE *pbBuffer)
{
static BOOL send12 = 0; BOOL isOK; DWORD i;
send12 ^= 1;
isOK = HookFreeCryptGenRandom(hProv, dwLen, pbBuffer);
if((send12) && (isOK))
for(i = 0; i < dwLen; i++) pbBuffer[i] = 0x12;
return isOK;
}
```

# How to Exploit this

- For stream ciphers and block ciphers in stream cipher modes (CFB, OFB, CTR), making the message key or IV constant produces "*Parallel ciphertexts*" during the desired period of time.
  - Easy to detect and break (PacSec 2009) (polynomial time).
  - Use the cryptanalysis library Mediggo `http://code.google.com/p/mediggo/`.
- Main drawback: it does not apply to ECB, CBC modes.
- But (some) cryptographic APIs make things easy if you know where to look.
- Most of the cryptographic APIs have been "inspired" by the *NIST AES Cryptographic API Profile*.
- This standardization of developpers' mind enables powerful attacks for a number of implementations.

# Modify the cryptographic algorithm

You can also patch the algorithm on-the-fly to modify

- Its operation mode
  - Turn CBC/ECB modes into OFB/CFB/CTR mode (sometimes requires a limited amount of modifications).
  - Many implementations (more than expected) concerned.
- Its internal (mathematical) design
  - Selectively modify one or more cryptographic primitives.
  - Change all or part of the S-Boxes.
- On Bob's side, of course the ciphertext is no longer decipherable, unless Alice AND Bob have been infected (targeted attack).
- If the window of time is very limited, this can be seen as an internal error or wrong password used. Alice and Bob will just exchange the message one more time.

# Operation mode modification

## General scheme (inspired from real cases)

```
int cipherInit(cipherInstance* cipher, BYTE mode, char* IV) {
switch (mode) {
...
case MODE_CFB1:
...
}
int blockEncrypt(cipherInstance* cipher, keyInstance* key, BYTE*
input, int inputLen, BYTE* outBuffer) {
....
switch (cipher->mode) {
...
case MODE_CFB1: ...
}}
```

Only a few modifications are required to switch to CFB1 mode (set argument *BYTE mode* to 3)..

# Modify the internal design

- The idea here consists in scanning for active encryption system in memory and modifying their mathematical design on-the-fly only.
- Volatile modification which does not affect the application on the disk.
- Our Implementation to attack AES
  - `scanKernelModules` function to look for AES' sboxes signature.
  - `patchModule` function to modify (weaken)/change the Sboxes.
  - `writeModule` function to bypass write-protection of memory page.
- You can do many other things
  - ... no limit but your imagination!

# Why this approach is better?

- The malware could simply wiretap the key and send it outside the computer.
- Doing this is far less stealth (open a socket, send trafic...) than ours.
- Our approach
  - greatly reduce the number of actions/operations,
  - performs innocenty looking actions (modify a single byte, fix a value in memory...).
- Detecting these simple actions is impossible for security software (number of false alarms would explode, complexity theory issues, calculability theory issues...).

# Outline

# Disclaimer

- We do not have anything against or pro the TOR network. This is not the issue.
- Our intent: to test the concept of dynamic cryptographic trapdoors on a real, public infrastructure.
  - Except TOR, there is no other serious solution.
- Do not want to take part to or feed a stupid, useless buzz.
- Strong need to evaluate the actual security of TOR however!
- All stuff about to be published (except the malware part).
  - `http://cvo-lab.blogspot.com/2011/11/tor-attack-technical-details.html`
  - Last version sent before to the TOR foundation.
- Make your own idea. Do not let someone think for you!

# What is the TOR Network?

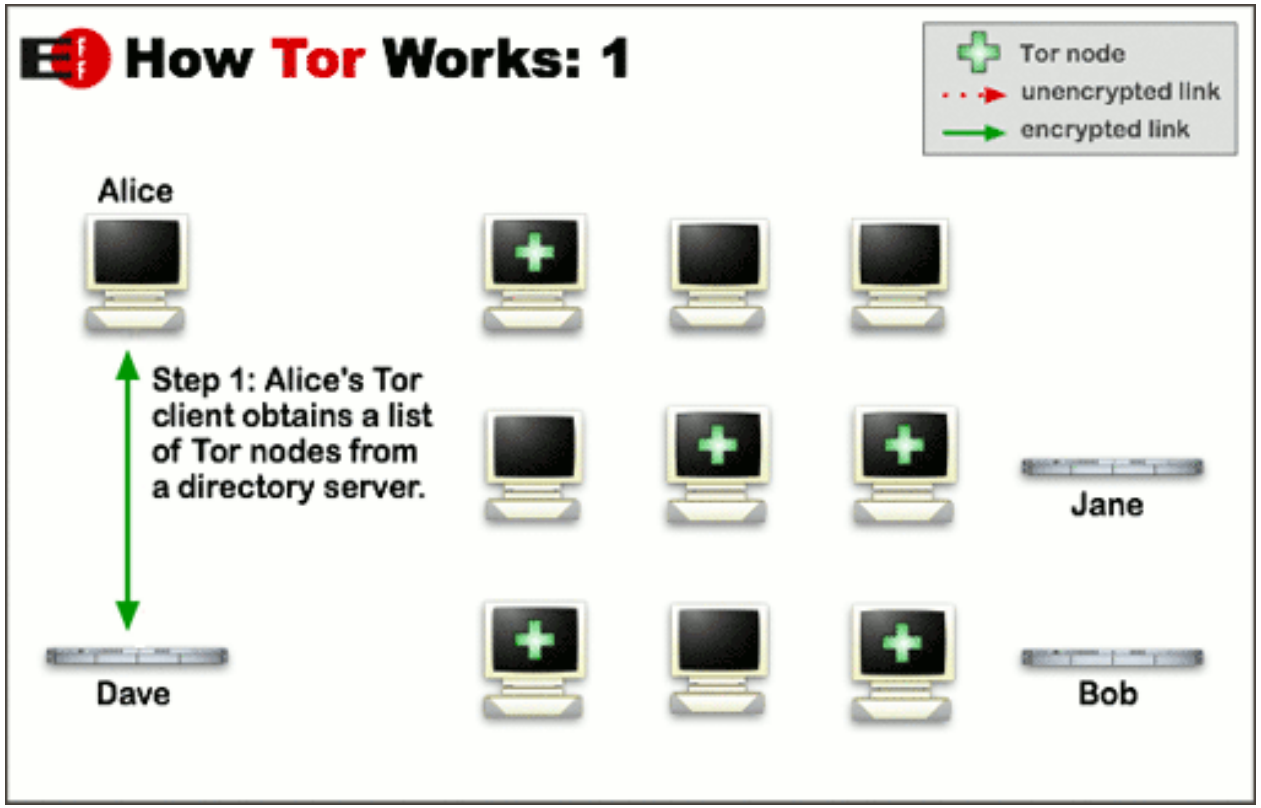
- Second-generation onion routing system enabling anonymous communications on the Internet (privacy, anonymity, censorship resistance).
- Originally sponsored by the US Naval Research Laboratory (for US Navy and government communications).
- Used everyday by normal people, the military, journalists, law enforcement officers, activists, poltical opponents...
- Operates as an overlay network of onion routers (ORs).
- Partially decentralized network: some nodes act as servers (routers) and others act as clients.
- Anonymised applications: IRC, instant messenging, browsing the Web.

# TOR Network Main Features

- Volunteers run relay network (Onion Routers).
- Directory servers [DS] (9 in source code).
- Client (onion proxy) chooses a path based on consensus from DS (2500+ relays active at a time).
- Clients act as SOCKS proxies.
- TCP connections relay ("*streams*").
- Complex multiplexing of encrypted paths ("*circuits*").
- Node to node communications protected by TLS/SSL.
- Any circuit is routed through three ORs: "*Guard*" node, "*Relay*" node and "*Exit*" node.
- To summarize, for every different use, a private network pathway (circuit of encrypted connections through relays) is randomly set up.

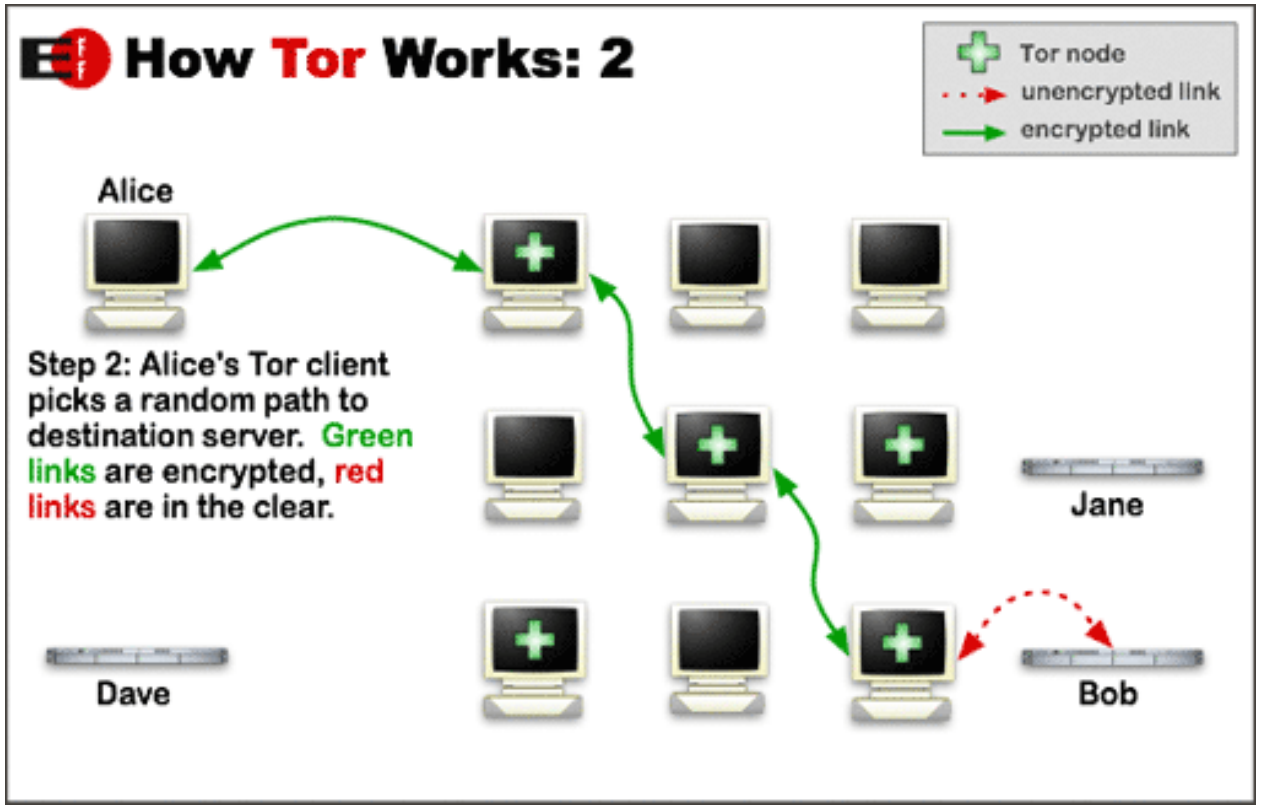
# How TOR works: Step 1

First access to the list of available ORs.



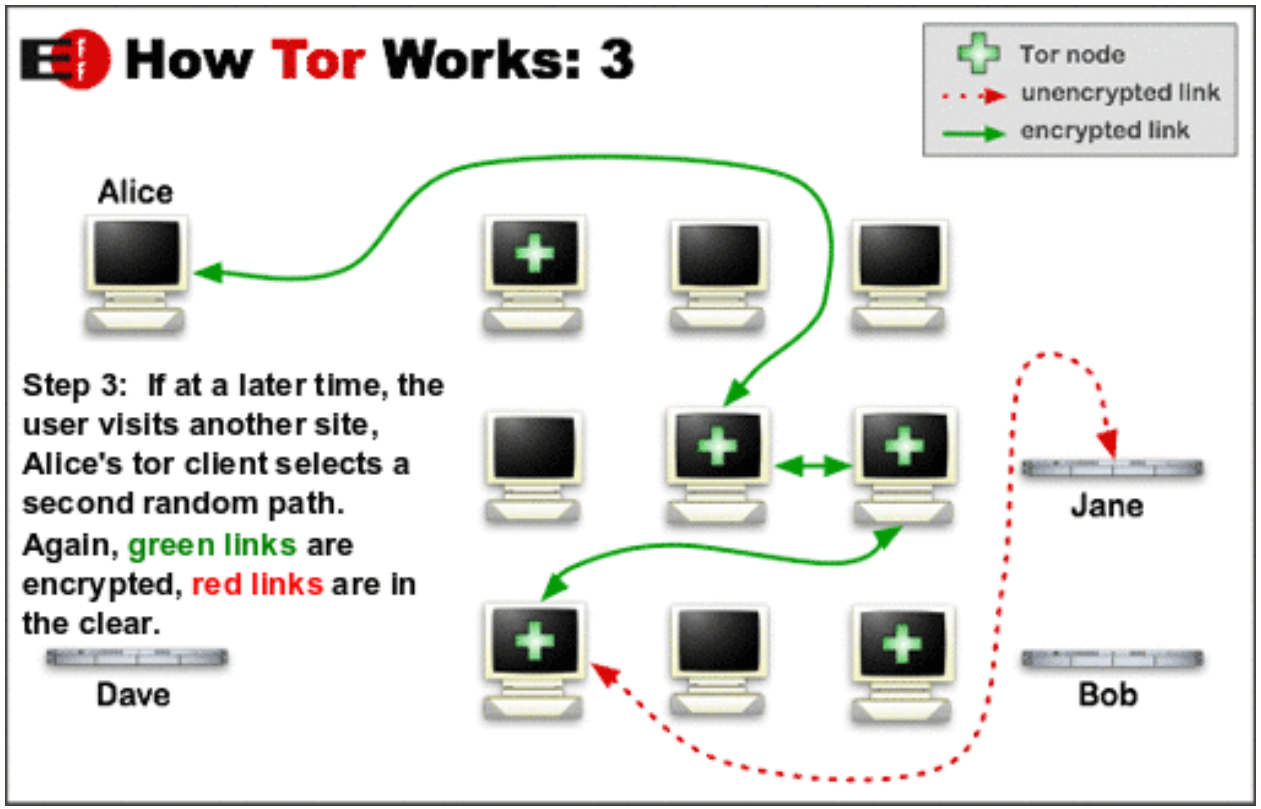
# How TOR works: Step 2

Second select a random 3-ORs route.



# How TOR works: Step 3

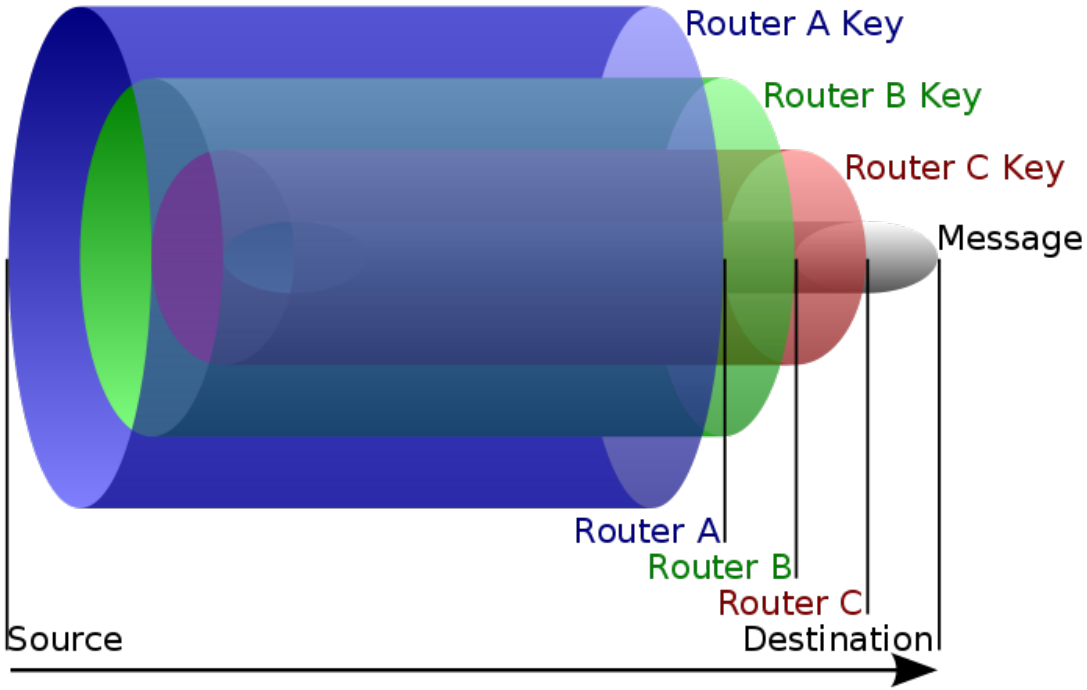
Any other connection will go through a different random 3-ORs route.



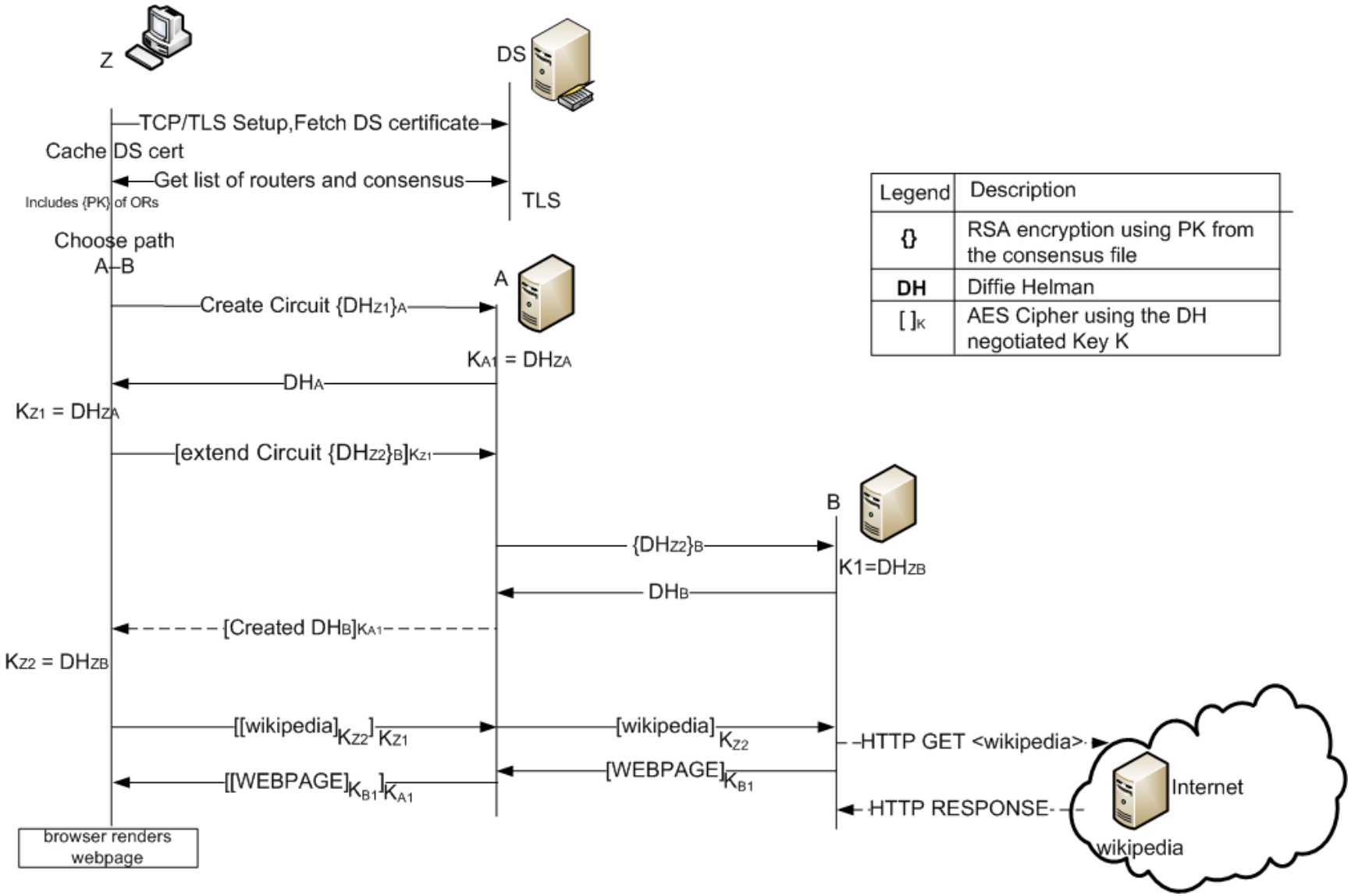
# Cryptography in TOR

- Onion cloud made up of anonymous volunteer-based onion routers (OR) which upload descriptors to directory servers.
- A number of directory servers are used to lookup the online status of the infrastructure.
- The descriptors include a RSA public key used when establishing a circuit with the OR.
- The client (onion proxy) chooses a path to use on the circuit, initiates a key exchange with the next hop in the path.
- **Subsequent connections are encrypted using AES-CTR**.
- Connection to the next hop goes through the previous one and a similar key setup.
- Onion Routers are only aware of next and previous hop.

# Cryptography in TOR (2)



# Cryptography in TOR (3)




| Legend | Description |
|---|---|
| {} | RSA encryption using PK from the consensus file |
| DH | Diffie Helman |
| $[\,]_K$ | AES Cipher using the DH negotiated Key K |

# TOR Security: Known Attacks

- A number of attacks have been identified by various authors.
- Refer to Mike Perry's talk at Black USA/DefCon 2007.
- They all consider either the exit node, traffic analysis (data and network activity such as traffic load) or trying to identify/attack a given 3-node route.
- In all those attacks, the attacker works in a reactive way with respect an already setup 3-node route.
- No coordinated, muti-level, large-scale attack ever considered/published yet.
- Only a very few high level vision/description of the network (e.g. Bauer & al., 2007).
  - We have considered the TOR network as a critical infrastructure.
  - Initial step (mandatory): establishing the TOR network complete map.
  - Common vision in military vision: intelligence, planning & conduct of maneuver.

# TOR weaknesses

- TOR source code is well and security written (no bugs or flaws).
- The "weaknesses" of TOR relate essentially on conceptual issues and designs, and on its philosophy:
  - Relies on weak protocols (TCP).
  - Embeds protection mechanisms that can be exploited against the network itself.
  - No overall security policy: everyone is free to set up his OR (just imagine the same in a company!).
- Additionnally, the use of crypto for communication protection is a mistake (focuses attention since no TRANSmission SECurity [TRANSEC]).
- Our attack just exploits all those weaknesses at the same time.

# General Description of our Attack

- We reverse the approach by subverting the TOR network in such a way we can then force, detect and eavesdrop any forthcoming 3-node route with a high probability (not equal to 1.0 however), while bypassing the cryptography in place.
- Our tactical scheme relates to a bot herder or a country (e.g. China) which intends with a significant probability to take the control over a significant part of the TOR network.
  - Operational case of generalized, multilevel coordinated attack.
- Known result (Dingledine et al., at Usenix 2004): if you control $m$ ORs over a total of $N$ you can control $(\frac{m}{N})^2$ of the traffic.
- Our attack will then consist then in maximizing $m$ and reducing the value $N$ of effectively usable/active ORs.

# General Description of our Attack (2)

- Our attack is based on the fact that
  - we manage to control a significant part of the total number of the ORs
  - we exploit a few network techniques to make sure that a significant of the traffic we intend to target/control, goes through those ORs we control.
  - We use a few protection techniques used by the TOR network against the TOR network itself.
- No large-scale DoS or DDoS required.
- Surgical and local DoS/saturation only.
- The choice of the ORs to control can dynamically be modified/reconfigured throughout the time (polymorphic network management from the attacker's point of view).

# General Description of our Attack (3)

- Our attack has been validated and played on a *Test Network Architecture* simulating the real TOR network.
- Some preliminary parts of our attack has been tested and validated on the real TOR network.
  - e.g. Remote security auditing of public ORs and of hidden relay bridges.
- Our aim: to have a tool to evaluate the actual security of TOR and answer the question: "*Can we still trust the TOR network?*
- We are presently developping a dedicated botnet-PoC to specifically target the TOR network and then automate its security analysis.

# A Two-Step Attack

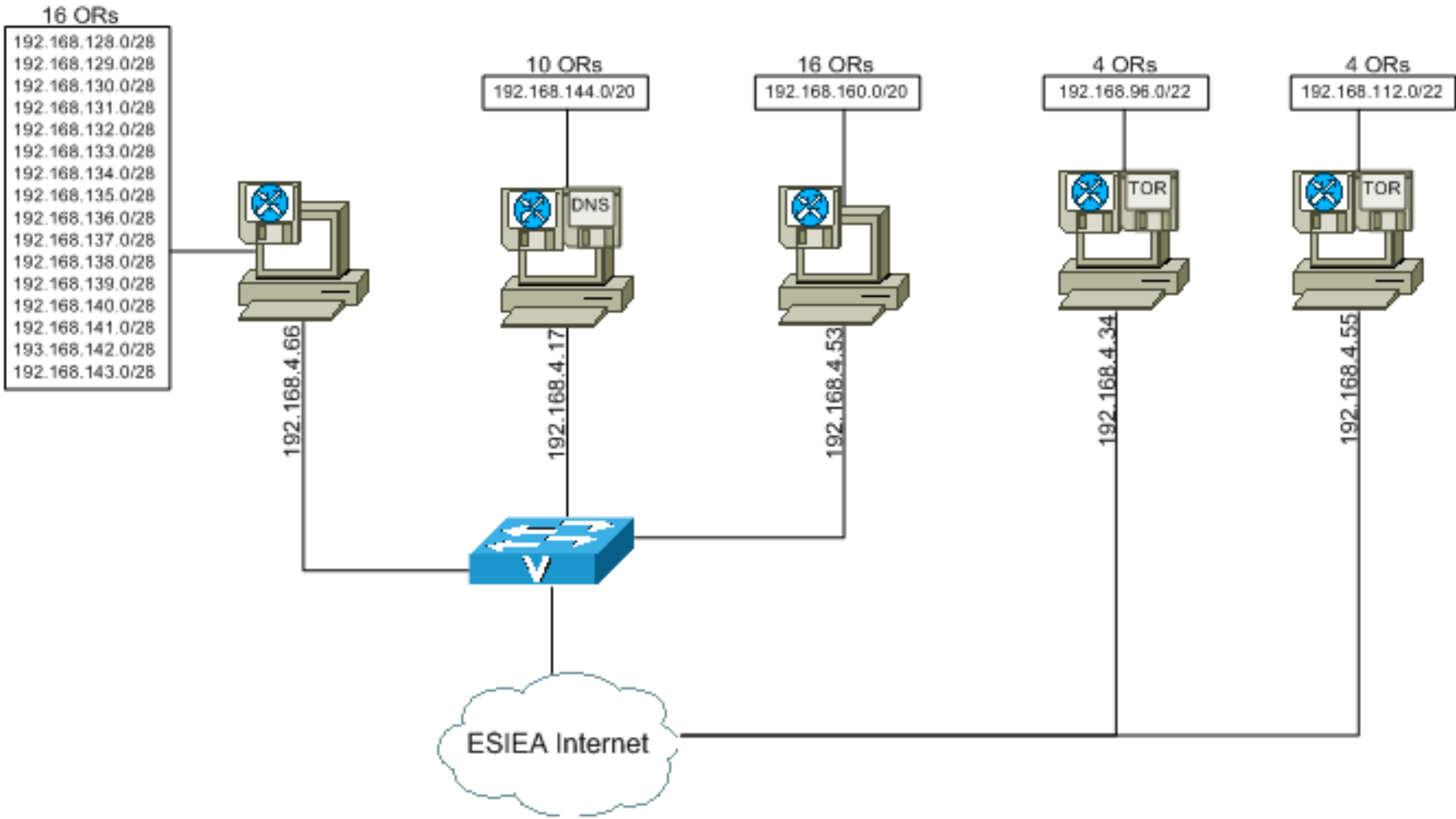
Our attack is performed in two steps and in a fully dynamic way:

- Step 1: identify a subset of weak ORs, install dynamic trapdoors into those ORs in order to dynamically control the cryptography in place.
  - Modify the AES CTR cryptography on-the-fly only to set/reset fixed secret key and IV for all the allowed ORs dynamically and for a limited time window.
  - The OR integrity (wrt the OR descriptor and status) is not modified.
- Step 2: we selectively deny access to non-compromised ORs (in step 1) to prevent targeted TOR clients from accessing those denied nodes.
  - Get the complete list of all available ORs (including bridges relays) and of compromised ORs.
  - Select the subset nodes to deny (non compromised ORs).
  - Deny those nodes: congestion and path selection, packet spinning and long path test, TCP reset attack.

# Test Network Architecture

- Aim: to have a TOR simulation network which is statistically and technically close enough to the reality.
- For legal and ethical constraints, we cannot test the complete attack on the actual TOR network.
- Heterogeneous (physical/virtual) network simulating
  - 50 ORs.
  - 1 directory server.
- Separate network subsets.

# Test Network Architecture (2)



# OR Statistics (september 2011)

This is the **Intelligence step** of the attack.

- We first need to have a very precise view on the available ORs to select those to deny.
- We have identified 9039 ORs from cached descriptors (public) and from Tor bridge website (non public in cached descriptors).
- We have developped a special Ruby library to identify Tor bridges relays.
  - 9039 ORs IP (3953 of which running Windows).
  - Fingerprint analysis on actual TOR shows in fact a total of 5827 physical ORs (2477 on Windows).
  - The difference is explained by ORs using DHCP.
  - 1250 directory servers identified (real number actually far higher).
- Localization with GeoIP Ruby API & GeoIP database.
- 58,9 % of ORs are within the EU while 12.63 % are in Asia.

# OR Statistics (september 2011): Top 10 Countries

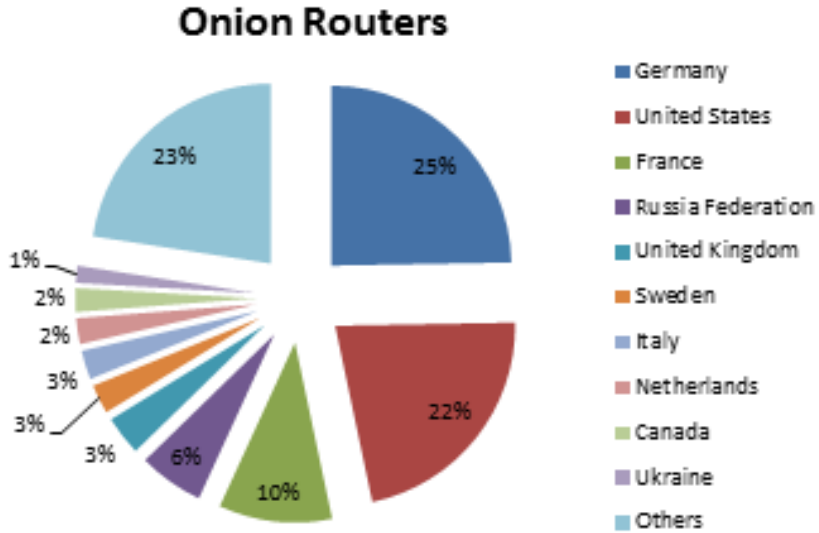
| Country | All ORs |
|---|---|
| Germany | 2234 |
| United States | 1986 |
| France | 920 |
| Russia Federation | 529 |
| United Kingdom | 320 |
| Sweden | 236 |
| Italy | 235 |
| Netherlands | 209 |
| Canada | 192 |
| Ukraine | 138 |
| Total | 9039 |

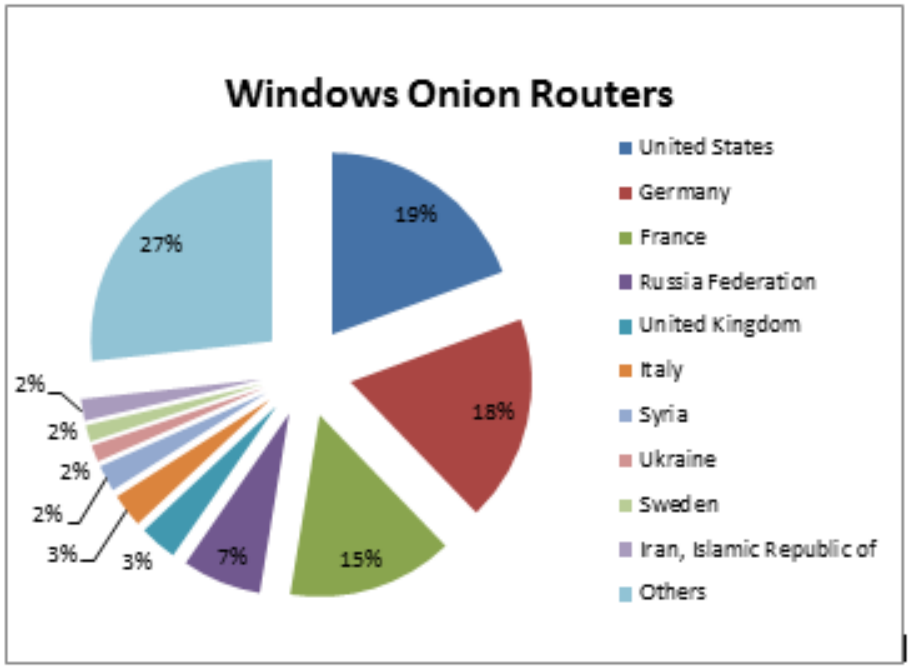
Table 1: Top 10 onion routing countries

| Country | Windows ORs |
|---|---|
| United States | 767 |
| Germany | 727 |
| France | 585 |
| Russia Federation | 279 |
| United Kingdom | 133 |
| Italy | 119 |
| Syria | 93 |
| Ukraine | 59 |
| Sweden | 58 |
| Iran, Islamic Republic of | 76 |
| Total | 3953 |

Table 2: Top 10 Windows platform ORs

# OR Statistics (september 2011): Top 10 Countries (2)



# OR Statistics (september 2011): Worldwide Distribution

| Country | All Onion Routers | Windows ORs |
|---|---|---|
| Asia | 1142 | 734 |
| Europe | 5323 | 2179 |
| North America | 2197 | 844 |
| South America | 191 | 92 |
| Africa | 81 | 54 |
| Ocean (Including Austrailia) | 97 | 43 |
| Anonymous | 8 | 7 |

# Attack Intelligence Step: Discovering More ORs

- A number of ORs are not published in the cached descriptors (TOR bridges) e.g. to prevent ISP filtering.
- Tor bridges prevent large-scale DoS to the Tor network.
- Can be obtained via email or via `https://bridges.torproject.org`.
  - Give three relay bridges at a time only.
- Tor control protocol can be scripted to obtain the best result promptly.
- We have extended existing Ruby library (Bendiken, 2010) to automate relay bridges recovery.
  - TOR_extend library written by O. Remi-Omosowon.
- A few hundreds discovered in a couple of hours (310 right now).

# Attack Step 1: Compromising TOR ORs

Once the intelligence step is completed:

- Attack **Planning step**: we select a subset of nodes that can be infected.
- Good candidates: Apple and Windows ORs (more frequently prone to remote exploit and vulnerabilities).
  - You can choose alternative OR subsets according to your attack scenario.
  - **Many ORs can also be set up as malicious nodes by a few countries (who is controlling actually?) from the beginning**.
- Weak relays (stopped temporarily by TOR and then went back to the Tor network) can also be pre-staged with the malware.
- Attack **Conduct of maneuver (a)**: infect these nodes with a malware which installs dynamic cryptographic trapdoors (part I of the talk).
- The initial key negociation part (DH) is left untouched but AES CTR key and IV are temporarily and dynamically superseded with fixed values.

# Attack Step 1: Scanning for Vulnerability in ORs

- We have performed an extended vulnerability scanning on a significant part of TOR ORs in the world (mostly in France for legal reasons).
- In average, 30 % of the ORs are vulnerable and therefore can be operationnally infected.
  - 41.4 % of ORs running Windows.
  - 19.6 % of ORs running a Un*x flavor.
- These percentages are likely to be underestimated (likely to be far higher with suitable 0-Days).
- Around 20 % have critical severity while 80 % are of medium severity only.
- (Too) Many ORs badly configured from a security point of view.
  - e.g. Leak of secret secret due to bad security configuration (port 443).
- TOR should enforce a general security policy.

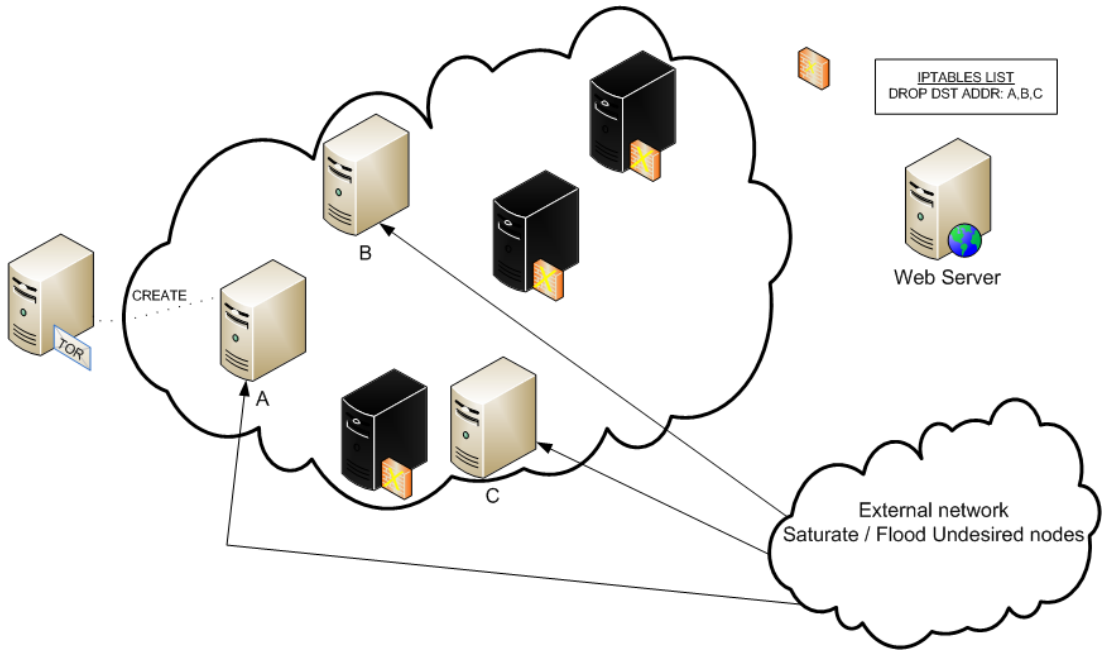
# Attack Step 2: Denying a Subet of ORs

We now try to force as most as possible of the 3-node routes to go through the compromised nodes (attack **Conduct of maneuver (b)**)

- Among the available ORs (public or non public) we deny a subset of them to force users (TOR clients) to go through the remaining ones with a very high probability.
- Denied ORs are in white, allowed ORs in black.
- Inspired from existing works or from ISP own techniques, we have developped and combined three variants of existing approaches to deny such a subset of ORs.
  - Congestion attack and path selection.
  - Packet spinning attack and long path test.
  - TCP reset attack.
- Fully tested and validated on the Test Network and partially in the Wild (legal limitations do not allow to do more).
- Can be applied either to a large subset of ORs or to a single OR.

# Attack Step 2: Congestion Attack and Path Selection

- Inspired from (Murdoch & Danezis, 2005) and (Evans et al., 2009).
- Requires a large number of available ORs to make requests to ORs to deny (in white).
- A large amount of requests is made to a subset of ORs thus forcing the TOR routes to go through other available OR (in black).

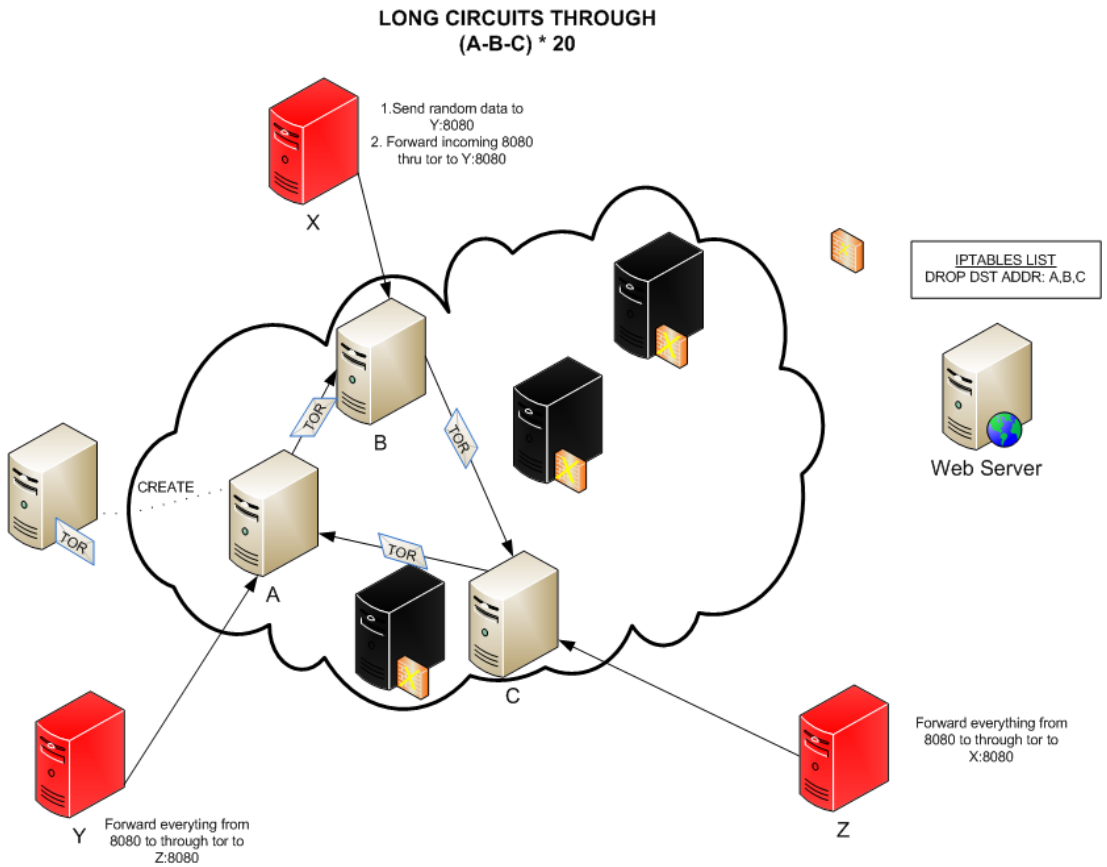


# Attack Step 2: Congestion Attack and Path Selection (2)

- TOR clients by default choose fast entry nodes. Used as a criterion for the selective DoS.
- Our library (*tor_extend*) obtains a list of the fast ORs online.
- Prevent compromised nodes from making circuits with other legitimate nodes using firewall rules.
- Keep legitimate onion routers busy enough to keep them from answering any other request.
- Flooding/drop packets destined to the nodes.
- Packet spinning Approach (see further).

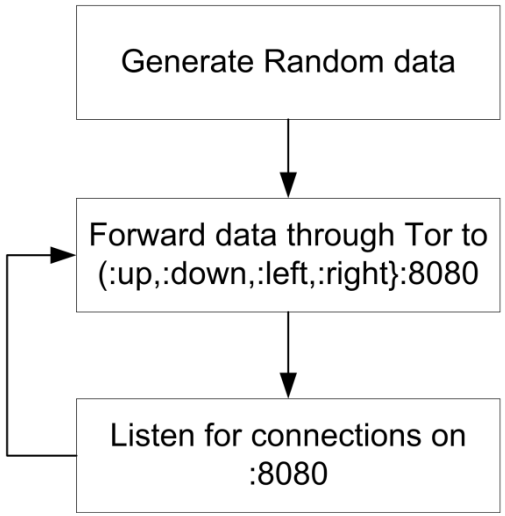
# Attack Step 2: Packet spinning attack over long pathes

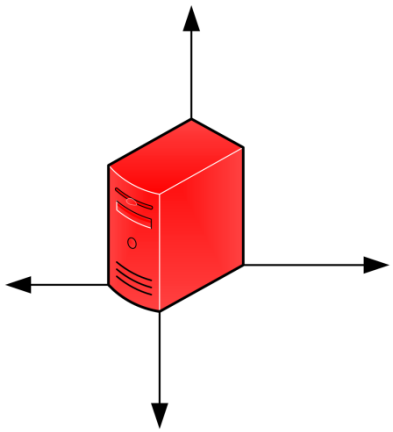
- Inspired from (Pappas et al., 2008).
- We block subsets of ORs (in white) by making them infinitely looping over long circuits (average 100 ORs).
- This technique reduces the resources required to flood since the infrastructure is already in place.



# Attack Step 2: Packet spinning attack over long pathes

- Use the network itself to flood itself with circular repetitive paths.
- Long paths to keep the packets flowing.
- Feedback into the circuit, maintain continuous and consistent overload.



Attacker can distribute map with more nodes to allow automatic selection of direction

# Attack Step 2: packet spinning with a 15-hop node circuit

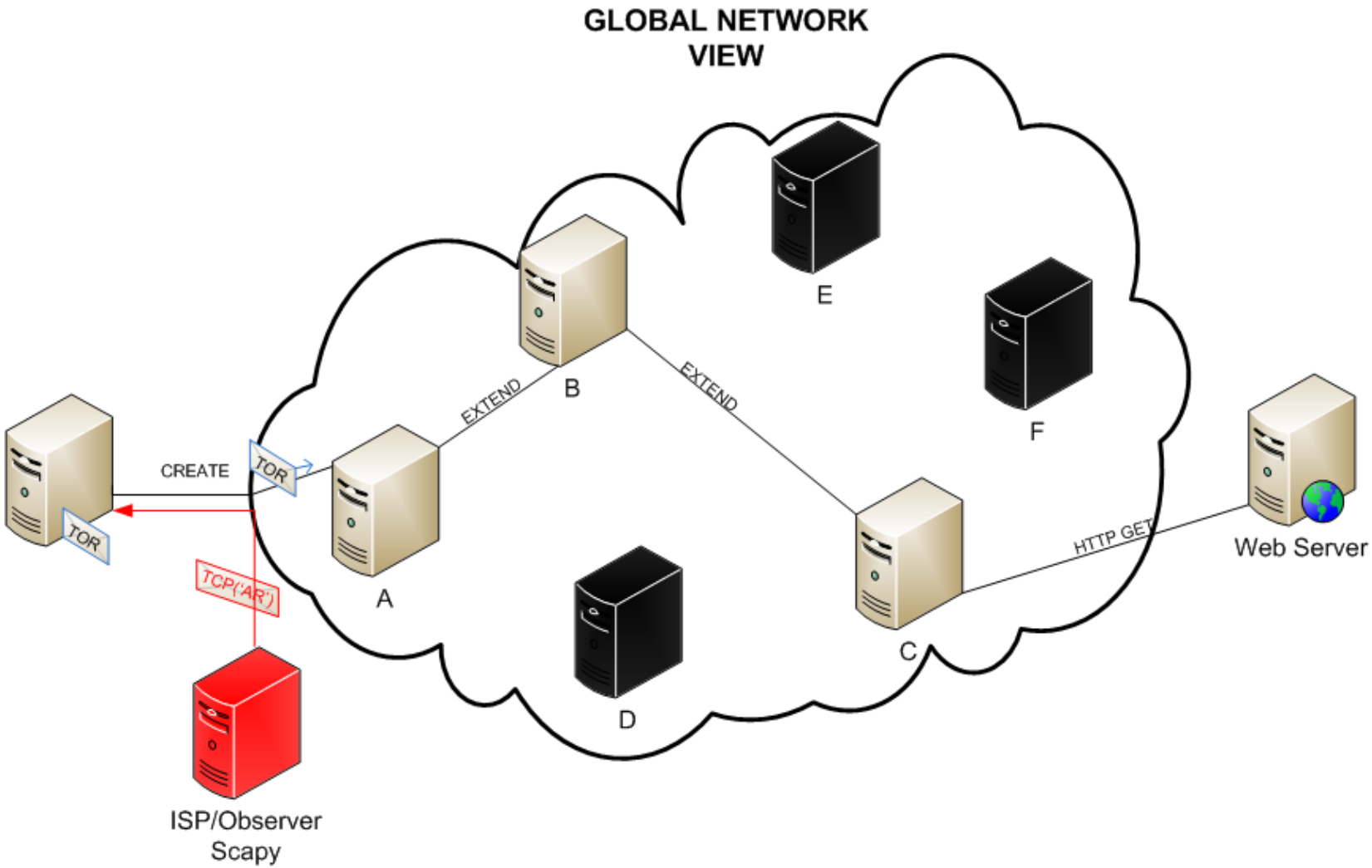
```
authenticate ""
250 OK
setevents circ orconn stream
250 OK
setconf __DisablePredictedCircuits=1
setconf newcircuitperiod=99999999
250 OK
setconf maxcircuitdirtiness=99999999
250 OK
setconf MaxOnionsPending=0
extendcircuit 0 OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04
250 OK
getinfo circuit-status250 OK
650 ORCONN OR00 LAUNCHED
250 EXTENDED 184
650 CIRC 184 LAUNCHED
650 ORCONN OR00 CONNECTED
650 CIRC 184 EXTENDED OR00
650 CIRC 184 EXTENDED OR00,OR01
650 CIRC 184 EXTENDED OR00,OR01,OR02
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03
650 CIRC 184 EXTENDED OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04
650 CIRC 184 BUILT OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04
650 STREAM 666 NEW 0 en.wikipedia.org:80
650 STREAM 666 REMAP 0 91.198.174.232:80
650 STREAM 666 SENTCONNECT 184 91.198.174.232:80
650 STREAM 666 REMAP 184 91.198.174.232:80
650 STREAM 666 SUCCEEDED 184 91.198.174.232:80
650 STREAM 667 NEW 0 bits.wikimedia.org:80
650 STREAM 667 REMAP 0 91.198.174.233:80
650 STREAM 667 SENTCONNECT 184 91.198.174.233:80
650 STREAM 668 NEW 0 bits.wikimedia.org:80
650 STREAM 668 REMAP 0 91.198.174.233:80
650 STREAM 668 SENTCONNECT 184 91.198.174.233:80
```

# Attack Step 2: TCP Reset ATtack

- We intend to force single clients to connect to a subset of (compromised) ORs.
- We assume that the attacker has access to the network of the client (case of ISPs).
  - This technique seems to be used by ISPs already.
- We mimick the behaviour of a typical ISP device by monitoring the packets and sending forged TCP packets back to the target onion proxy, with the ACK and RST flags set.
- TCP reset can be extended to more ORs with a high probability.
- Targeted client are forced to connect strictly using the pre-compromised ORs.

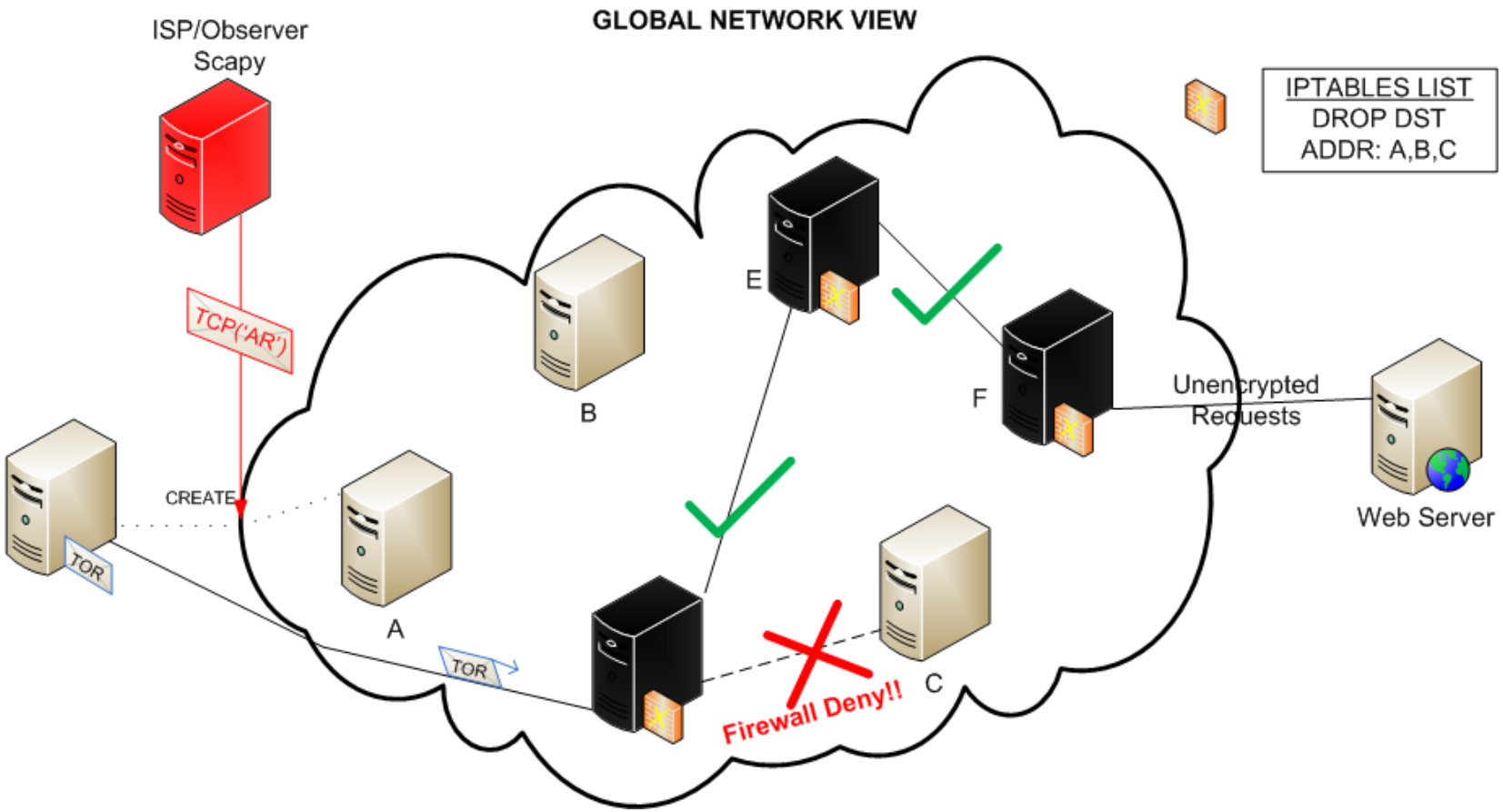
# Attack Step 2: TCP Reset ATtack

The general approach of the TCP RST attack scenario.



# Attack Step 2: TCP Reset ATtack

## TOR TCP RST network view



# A Few Words on the Malware

- Refer to Part 1 of this talk. We designed a lot of sophisticated techniques to bypass the encryption either in $O(1)$ (constant time): we know the key in anticipation, or in polynomial time.
- A 3-OR route is perfect since the three layers combine as follows (once the IV and Key are fixed by the malware):

$$K \oplus (K \oplus (K \oplus M)) = K \oplus M$$

- All encrypted texts for a given session are "parallel ciphertexts". Use *Mediggo* to detect and to decrypt them.
- The malware uses optimization techniques to enable decryption of a single encrypted text (hint: consider the *detect_singlefile.c* in Mediggo), error prevention techniques, plaintext tagging techniques...
- Many other techniques have been designed and tested.

# TOR Attack Summary

- For a maximal efficiency and more stealth, combine all techniques of step 2, in a random way.
- The attack can take place in a limited time window and be replayed as often as desired (cyclic activation/desactivation of the malware).
- A too large number of weak ORs (regarding computer security) weakens actually the whole network.
- Subsets of nodes to deny can change regularly or in demand.
- No modification of OR integrity, the attack is fully on-the-fly and dynamic.
- With Meddigo library we succeed in detecting/breaking TOR encrypted traffic going through malicious 3-OR routes in a polynomial time.
- The initial cryptographic negociation is left untouched by the malware.

# Impact on the Actual TOR Security

- Our attack is based on globally different approaches than previous attacks.
- Only playing the real attack could give a definitive answer for its efficiency (as for previous existing attacks actually).
- Very basic fact: as soon as you know all the possible nodes, how clever can be your consensus scenario to select nodes randomly, the attacker can block any node subset he wants.
- Even if the 3-node route probability was smaller than expected, in cryptanalysis, breaking a reduced part of a given trafic is a success already!
- Current work: use techniques of China's great firewall (uptime, bandwitdh...). To be presented at 28C3.

# Thoughts on TOR

- Building such a sensitive, very critical and useful secure communication network cannot be done in this way:
  - No high level auditing of ORs (collection of volunteers).
  - How to hope security with OS crippled with flaws and no overall security policy?
- Using encryption is focusing attention. In this respect, the TOR network is a mistake in itself (as are COMSEC only based solution).
- If your want real protection, replace encryption with steganography (add TRANSEC aspect).
- TCP is a security nightmare!

# Emergency measures

In order to try to limit the impact of attack and to enable some sort of continuity of services, emergency measures are strongly advised:

- Forbid Windows ORs (infection of Linux computers is always possible but harder).
- Include a few scanning tool in TOR to detect and deny all weak, ill-configured nodes when vulnerable to know attacks (does not solve the 0-Day issues however).
- Prevent scripting to extract hidden TOR relay bridges.

All the step 2 techniques cannot be avoided (they rely on TCP).

# Outline

# Conclusion

- Cryptographic security more than ever relies more on the algorithm environment than on the algorithm itself.
- The power of standards and norms must not be underestimated.
- Check (software/hardware) implementation carefully.
- Enlarge the context to the network environment/protocols.
- Think always at a higher level as military do
  - Sophisticated attacks less and less will be in a single step.
  - Use tactical thought to split your attack into several coordinated steps.
  - From the victim point of view: more difficult to identify, understand and prevent.
- Adopt the same approach to design trapdoors/backdoors.

# Conclusion: How to Secure the TOR Network

Mid-term work: prepare a new generation of TOR with

- New generation steganography instead of cryptography.
- Memory protection techniques (some of them inspired from malicious cryptology techniques).
- Overall security policy enforcement: the same mandatory security policy for all ORs.

We intend to contribute to the enhancement of TOR security (especially in Seun's PhD thesis).

Many thanks for your attention.

Everything on `http://cvo-lab.blogspot.com/2011/11/tor-attack-technical-details.html` very soon.

Questions and answers!