Dynamic Cryptographic Backdoors Part II
Taking Control over the TOR Network

Eric Filiol (speaker) - Oluwaseun Remi-Omosowon (speaker) - Leonard Mutembei

filiol@esiea.fr, seunomosowon@gmail.com

http://sites.google.com/site/ericfiliol

https://sites.google.com/site/esieanismaster/

ESIEA - Laval
Operational Cryptology and Virology Lab (C + V)^{o}

28C3 2011 - Berlin

Outline

1 Introduction

2 Dynamic cryptographic trapdoors
- Introduction
- OS level dynamic trapdoors
- Algorithm level dynamic trapdoor

3 Taking Over the Tor network
- Tor network description
- Cryptography and security in Tor network
- Taking control over the Tor network

4 Conclusion

Cryptanalysis reality

- What does “to break cryptography” mean?
- Use the “armoured door on a paper/cardboard wall“ syndrome?
  - 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.
- Can we subcontract security stuff to official organizations (GOs or NGOs)?
- Think in a different way and far from the official cryptographic thought.
- To break a system means actually and quickly access the plaintext whatever may be the method.

Dynamic Cryptographic Backdoors Part 1 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.
  - All techniques tested and validated in real conditions/environments.
- Let us now present how to use all of this to take control over the TOR network in a dynamic way.
- 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.

Malware

- We all know what a malware is
- Electronic Frontier Foundation (https://ssd.eff.org/tech/malware)
    - "The risk that any given computer is infected with malware is therefore quite high unless skilled computer security specialists are putting a substantial amount of effort into securing the system."
    - "It is unlikely that U.S. government agencies would use malware except as part of significant and expensive investigations"
- Problem:
    - We think attackers are one step behind
    - Will governments bother with traffic confirmation if they have no access to the destination server?
    - Military == Coordinated significant attacks
- Operational fact:
    - Accessing 1% of plaintext is already a cryptanalysis success!

Summary of the talk

1 Introduction

2 Dynamic cryptographic trapdoors

Recall of previous chapters (CanSecWest 2011 mostly)

3 Taking Over the Tor network

Tor network description

Cryptography and security in Tor network

Taking control over the Tor network

4 Conclusion

Outline

1 Introduction

2 Dynamic cryptographic trapdoors

- Recall of previous chapters (CanSecWest 2011 mostly)

3 Taking Over the Tor network

- Tor network description
- Cryptography and security in Tor network
- Taking control over the Tor network

4 Conclusion

Recap: Dynamic Cryptographic trapdoor

- We examine how a simple malware can be used for coordinated attack
- Many encryption algorithms rely on the operating system primitives to generate the IVs and secret keys (e.g. Microsoft cryptographic API).
  - Hook the API function
- Cryptographic algorithms can be modified in memory: mode/design
  - No modification on the hard disk (no static forensics evidence).
  - Turn CBC/ECB modes into OFB/CFB/CTR mode
- The trapdoor has a limited period of time and can be replayed more than once. Dynamic periods of time with weak encryption.
- The attacker has just to intercept the ciphertext and perform the cryptanalysis in polynomial time.
- The “static (mathematical) security” remains unquestioned!
- Same approach for other equivalent resources (network infrastructure, key infrastructure, network-based key management...)

Recap: Hooking the CryptGenRandom function

- A malicious DLL is injected in some (suitable) processes. This DLL hooks the *CryptGenRandom* function (included in *Microsoft's Cryptographic Application Programming Interface*).

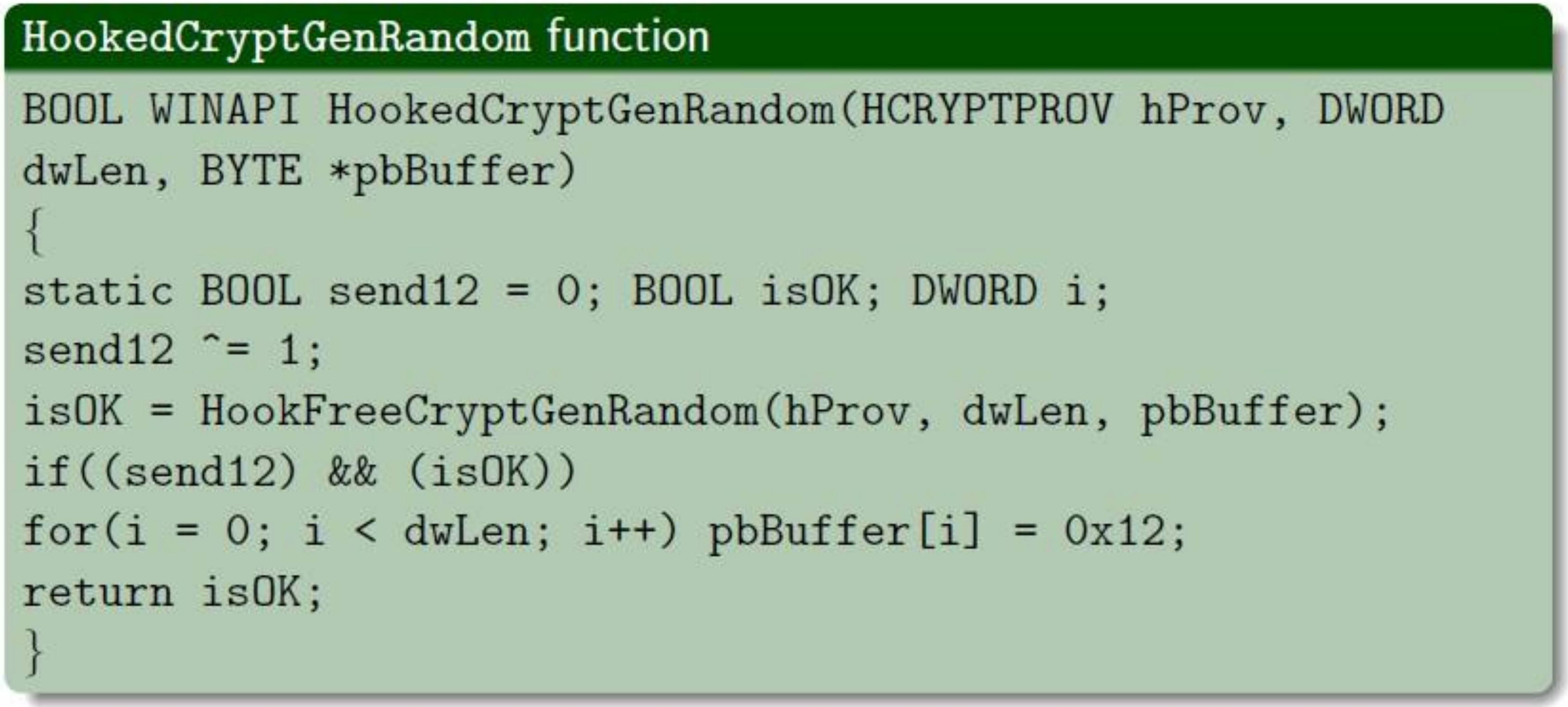
CryptGenRandom function

```
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 2 pm only.
- *CryptGenRandom* return value is modified by the function *HookedCryptGenRandom* (fixed value).
- On Bob's side, the cipher text can still be deciphered.

Recap: Hooking the CryptGenRandom function (2)

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;
}
```

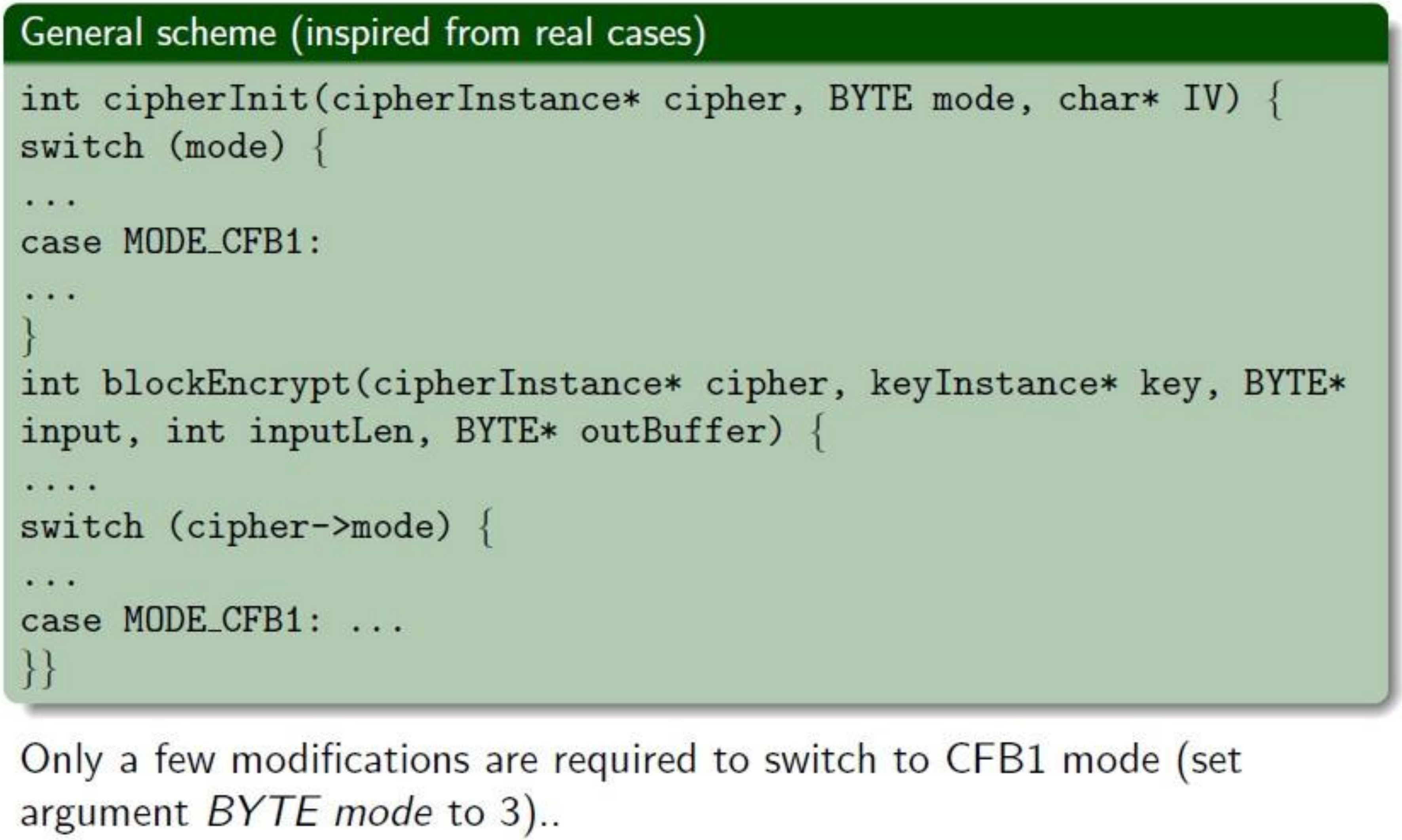
Recap: cryptanalysis step

- For stream ciphers and block ciphers in stream cipher modes (CFB, OFB, CTR), making the message key or IV constant produces “*Parallel ciphertexts*” over the chosen time window.
  - Easy to detect and break (PacSec 2009 - Black Hat Europe 2010) (polynomial time).
  - Use the cryptanalysis library Mediggo http://code.google.com/p/mediggo/
- It does not apply to ECB, CBC modes.
- But (some) cryptographic APIs make things easy if you know where to look.
- Most cryptographic APIs have been “inspired" by the *NIST AES Cryptographic API Profile*.
- This standardization of developers' mind enables powerful attacks for a number of implementations.

Recap: Modify the cryptographic algorithm

- You can also patch the algorithm on-the-fly to modify
  - Its operation mode (many implementations concerned).
  - Its internal (mathematical) design
    - Selectively modify one or more Boolean functions
    - Change all or part of the S-Boxes.
- The idea here consists in scanning for active encryption system in memory and modifying their mathematical design on-the-fly only.
- On Bob's side, of course the cipher text 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.

Recap: Modify the cryptographic algorithm (2)



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)..

Outline

1 Introduction

2 Dynamic cryptographic trapdoors

- Recapitulate

3 Taking Over the Tor network

- Tor network description
- Cryptography and security in Tor network
- Taking control over the Tor network

4 Conclusion

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 public solution.
- Do not want to take part to or feed a stupid, useless buzz.
- Strong need to evaluate the actual security of TOR however!
- Details published online (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!
- **Strong point: our attack considers the TOR network as a critical infrastructure. We do not evaluate the TOR technology in itself but we mostly exploit its weak deployment by volunteers and its use of weak protocols (TCP)!**

What is the TOR Network?

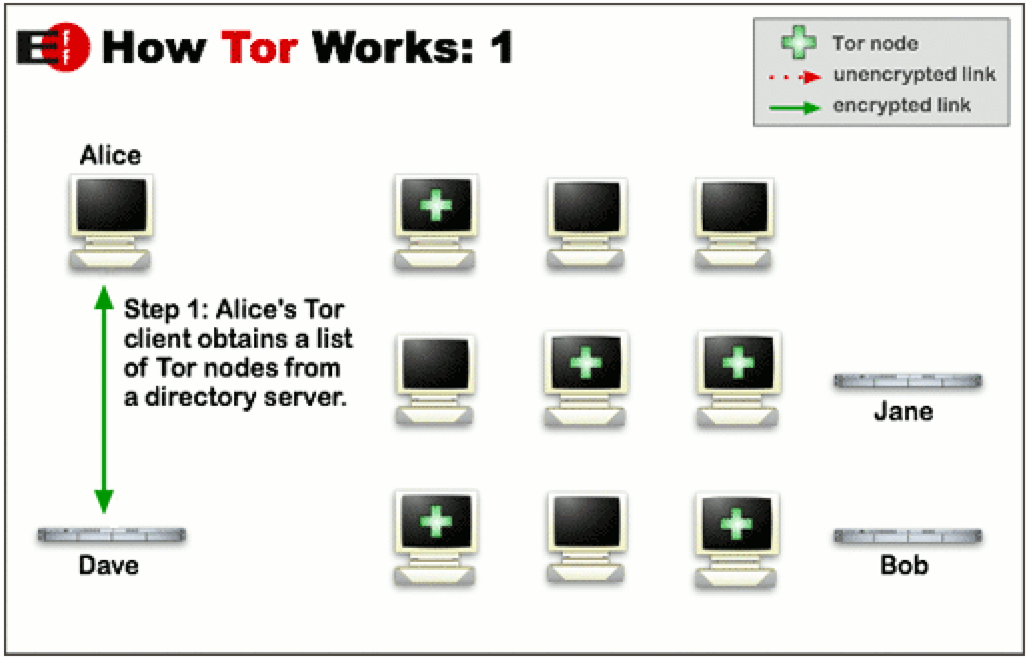
- Second-generation onion routing system enabling anonymous communications on the Internet (privacy, anonymity, censorship resistance).
- Originally sponsored by the US Naval Research Laboratory (for US Navy and government communications). Now an Electronic Frontier Foundation (EFF) project (2004).
- Used everyday by normal people, the military, journalists, law enforcement officers, activists, political 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 messaging, browsing the Web.

TOR Network Main Features

- Volunteer run relay network (Onion Routers).
- Directory servers [DH] (9 in source code), optional on volunteer relays.
- Client (onion proxy) chooses a path based on consensus from DS
- Metrics used based on bandwidth, uptime, OS
- Clients act as SOCKS proxies.
- TCP connections relay (“streams”).
- Complex multiplexing of encrypted paths (“circuits”).
- Node to node communications protected by TLS/SSL.
- circuits made up of “Guard” node, “Relay” node and “Exit” node.
- To summarize, a private network pathway (circuit of encrypted connections through relays) is randomly set up that supports various applications

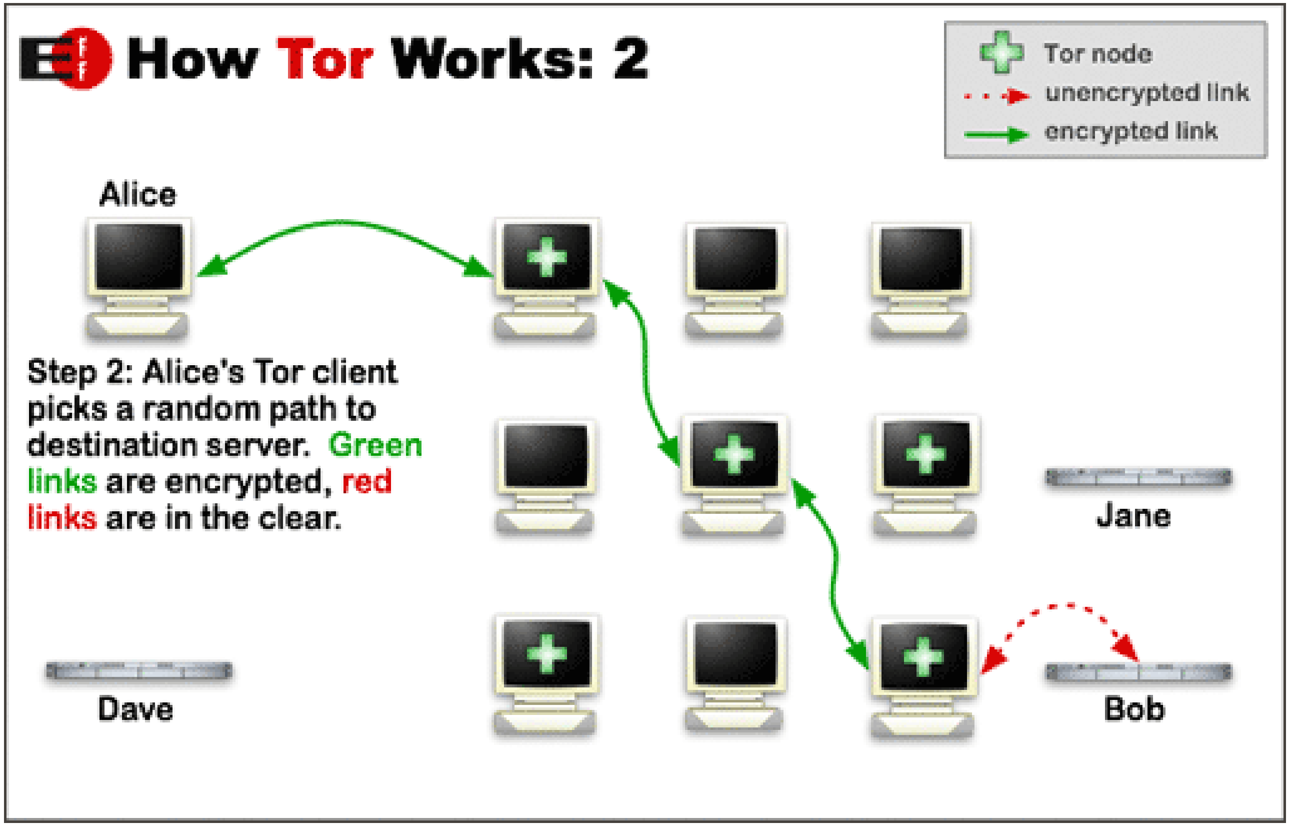
How TOR works: Step 1

- First, access to a list of available ORs - descriptors
- Choose randomly and create circuit
  - relay with guard flag and high metric (fast, high uptime)
  - Suitable relay
  - Suitable exit node with ‘exit’ flag and no access restriction to desired port/service


How Tor Works: 1
Tor node
unencrypted link
encrypted link
Alice
Step 1: Alice's Tor client obtains a list of Tor nodes from a directory server.
Jane
Dave
Bob

(Source EFF)

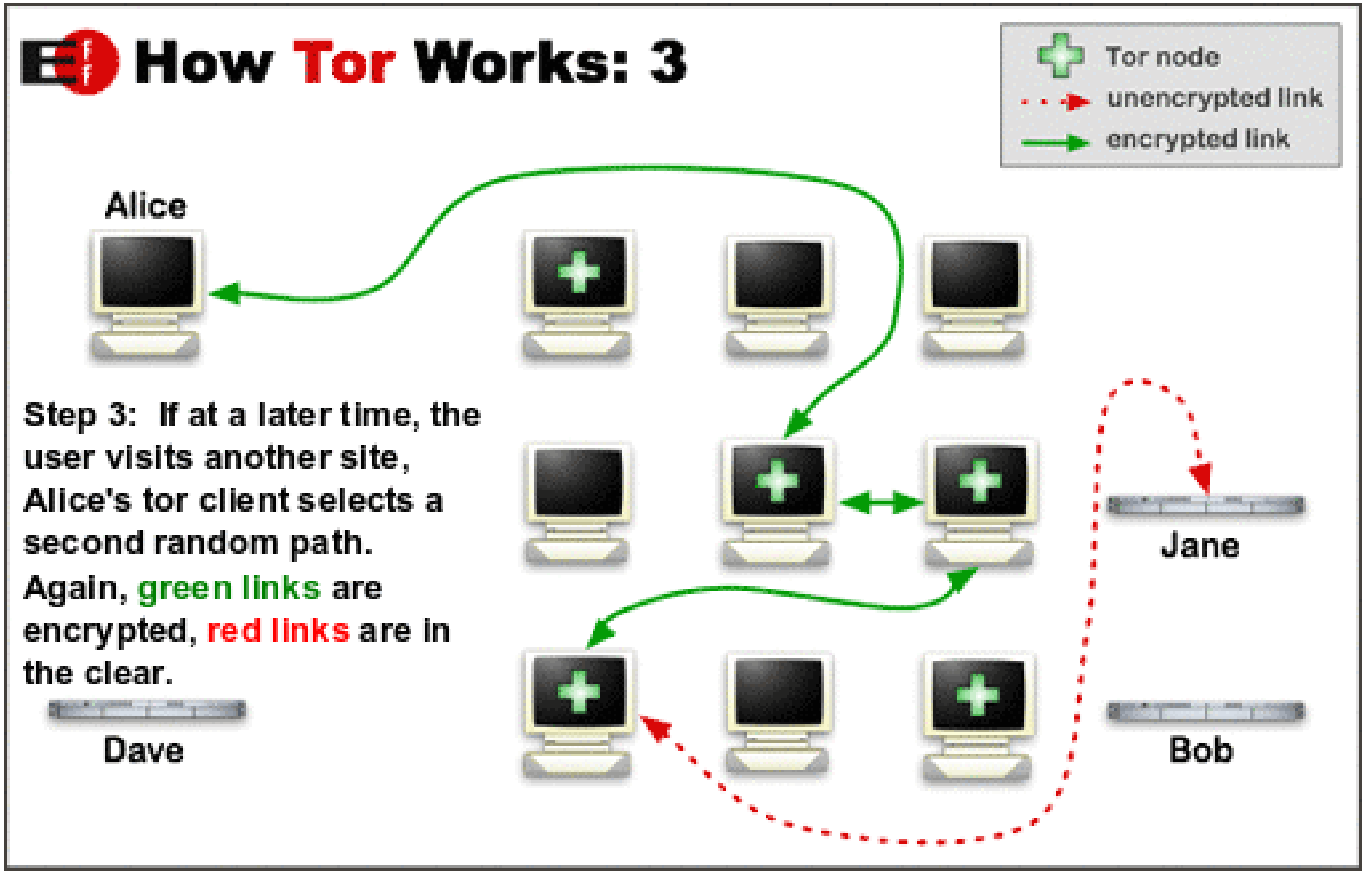
How TOR works: Step 1



Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.

(Source EFF)

How TOR works: Step 1

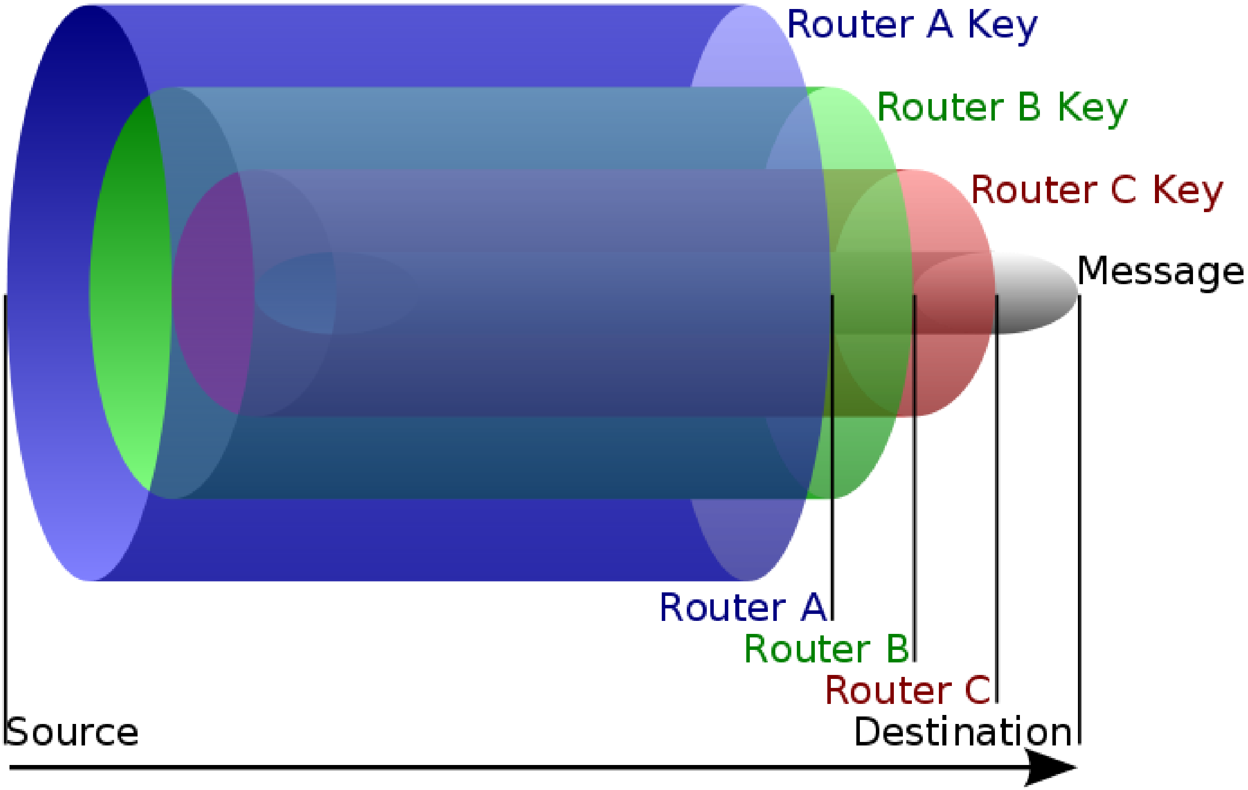

How Tor Works: 3
Tor node
unencrypted link
encrypted link
Alice
Step 3: If at a later time, the user visits another site, Alice's tor client selects a second random path. Again, green links are encrypted, red links are in the clear.
Jane
Dave
Bob

(Source EFF)

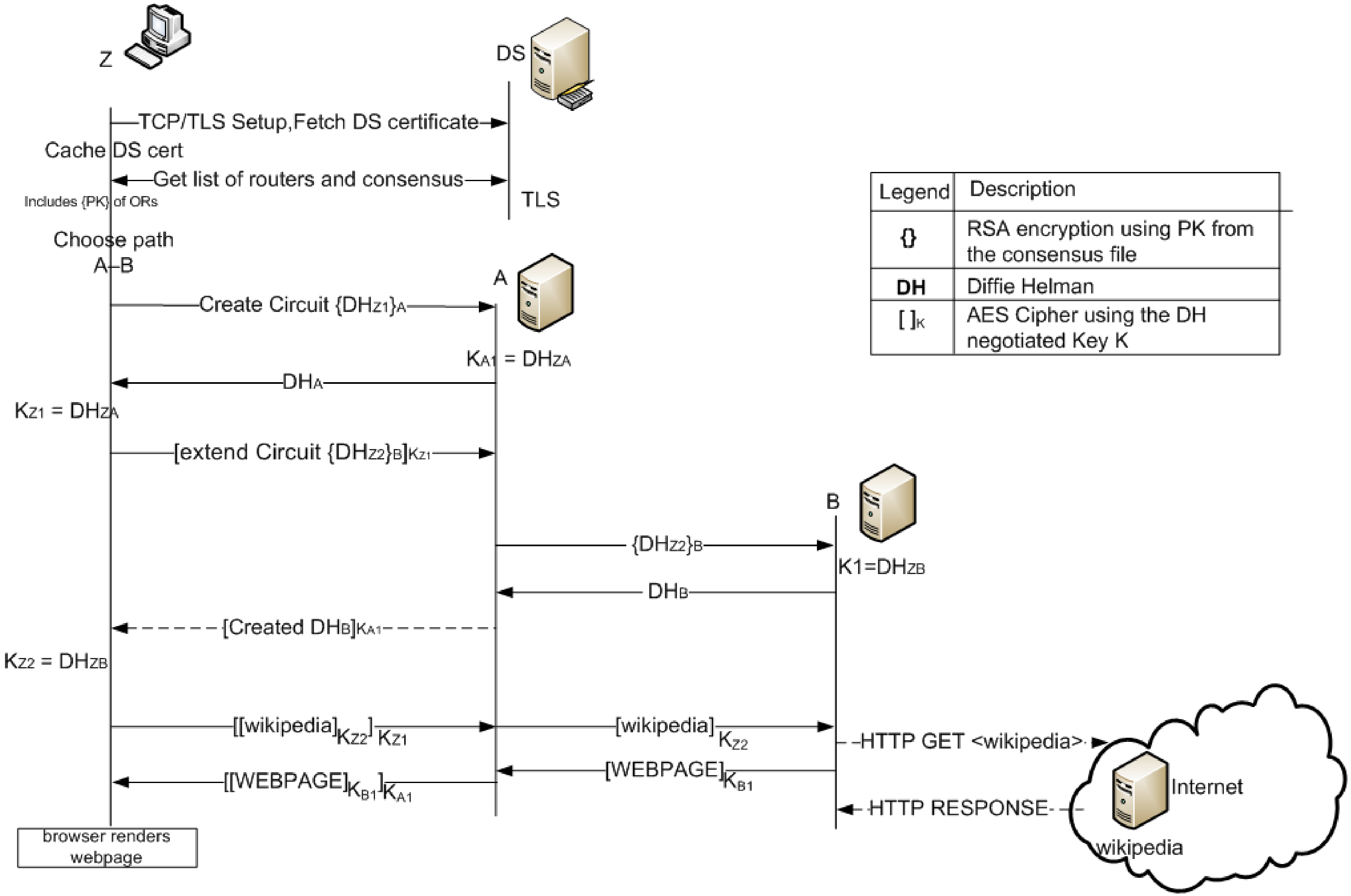
Cryptography in TOR

- Multiple streams can use the same circuit
- Onion cloud made up of anonymous volunteer-based onion routers (OR) which upload descriptors to directory servers.
- Directory servers are used to lookup online status of the infrastructure.
- Each descriptor includes RSA public key of 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.**
- Similar key setup Connection to the next hop.
- Onion Routers are only aware of next and previous hops.

Cryptography in TOR (2)



Cryptography in TOR (3)



Z

DS

TCP/TLS Setup,Fetch DS certificate

Cache DS cert

Get list of routers and consensus

Includes {PK} of ORs

TLS

Choose path A-B

A

Create Circuit $\{DH_{Z1}\}_A$

$K_{A1} = DH_{ZA}$

DH_A

$K_{Z1} = DH_{ZA}$

[extend Circuit $\{DH_{Z2}\}_B]_{K_{Z1}}$

B

$\{DH_{Z2}\}_B$

$K1=DH_{ZB}$

DH_B

[Created $DH_B]_{K_{A1}}$

$K_{Z2} = DH_{ZB}$

$[[\text{wikipedia}]_{K_{Z2}}]_{K_{Z1}}$

$[\text{wikipedia}]_{K_{Z2}}$

HTTP GET <wikipedia>

$[[\text{WEBPAGE}]_{K_{B1}}]_{K_{A1}}$

$[\text{WEBPAGE}]_{K_{B1}}$

HTTP RESPONSE

Internet

wikipedia

browser renders webpage

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, large-scale attack ever considered/published yet.
- Only a very few high level vision/description of the network (e.g.
- Bauer et 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 manoeuvre.

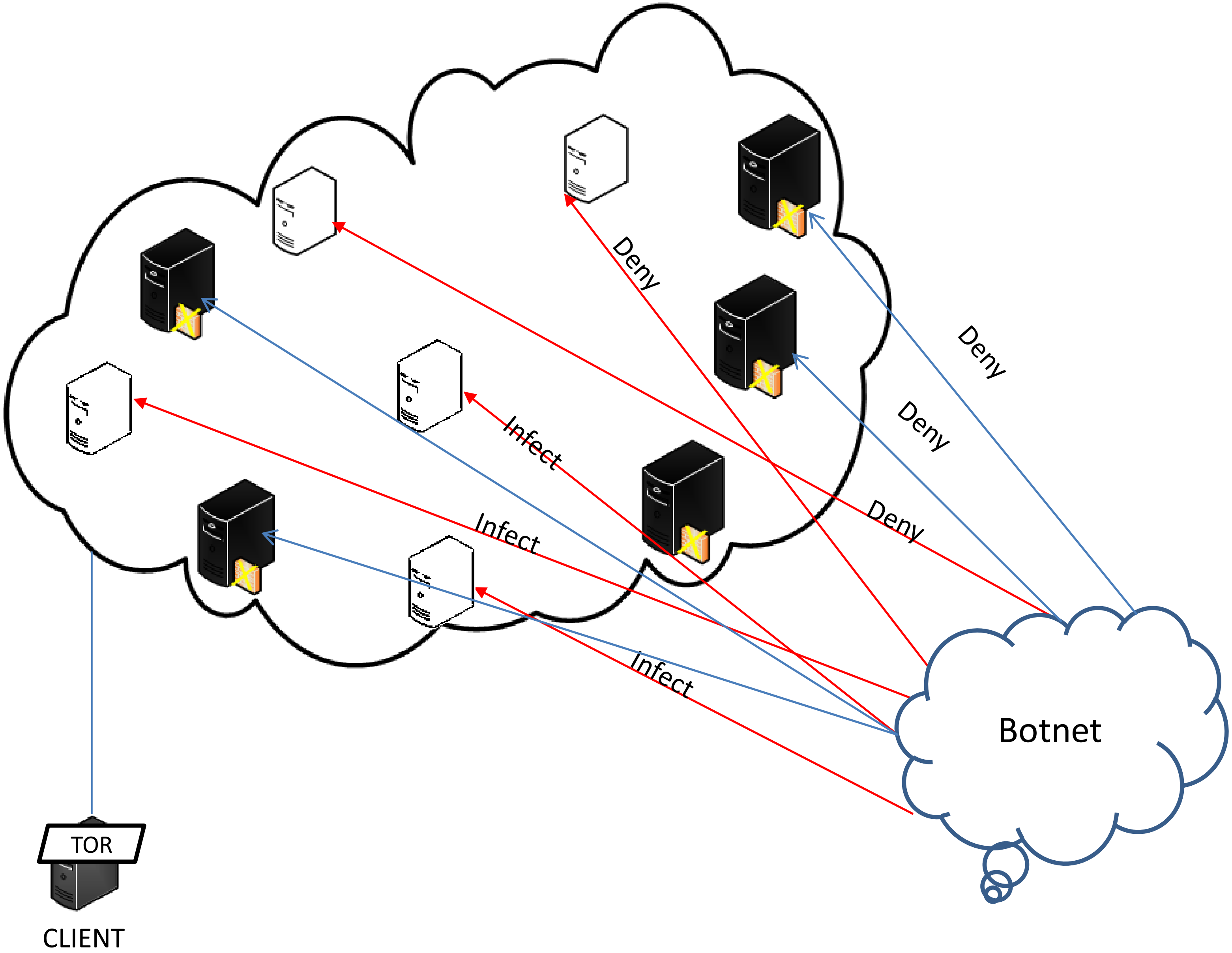
TOR Weakness

- TOR source code is well and securely written (no bugs or flaws).
- The core concept is sound and elegant.
- The “weaknesses” of TOR relates 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!).
- Additionally, 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 overview of the attack



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
  - Make sure that most of the traffic we intend to target/control, goes through those ORs we control.
  - We use the Tor network against itself
- No large-scale DoS or DDoS needed, 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).
- **Attack validated on a *Test* Network Architecture simulating the real one.**
- Critical parts of our attack have been tested on the real TOR network.
- We are presently developing a dedicated botnet-Poc to evaluate and analyze the Tor security on the real network and try to answer the question: "*Can we trust the TOR network?*"

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.

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 over 9039 ORs from cached descriptors (public) and 355 bridge addresses from Tor bridge website (non public in cached descriptors).
- We have developed 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 (2487 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.

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 controller (Bendiken, 2010) to automate bridge enumeration.
  - *tor_extend* library written by O. Remi-Omosowon.
- Demo 1

Attack Intelligence Step: Discovering More Ors(2)

- Obtain bridges from webpage promptly using multiple Tor exit nodes
- http://tor-extend.rubyforge.org/
- Demo 2

```
C = torct.get_purposeip("exit fast")
url = URI.parse "https://bridges.torproject.org/"
C.count.times{|z|
   circuit1 = [entry,relay, C[z] ]
   cir_num = mytor.extendcir(0, circuit1)
   if mytor.cir_status.detect{|p| p =~ /#{cir_num} BUILT/}
      bridgelist |= mytor.get_bridges(url,cachedf)
   end
   mytor.closecir(cir_num)
}
```

- 70 distinct bridge addresses in 10 minutes
- Hundreds in 1 hour (200+)
- Requests can be multi-threaded to use half the time
- Recent similar approach to ours by Ling et al (2011) detection of 2300+ bridges from emails and bridge website using PlanetLab nodes and confirmed existence of over 10000 Tor bridges (http://www.cs.uml.edu/~xinwenfu/paper/Bridge.pdf)
- Easy for script kiddies
- 355 in Google earth file, but easy to enumerate.

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.
  - **Several ORs can also be set up by a few countries without the need to compromise.**
- Weak relays (stopped temporarily by TOR and then went back to the Tor network) can also be pre-staged with the malware.
- Infect selected nodes with malware (dynamic cryptographic trapdoor).
- The initial key negotiation 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 operationally infected.
    - 41.4 % of ORs running Windows.
    - 19.6 % of ORs running a Un*x flavour.
- 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.
- TOR should enforce a general security policy.

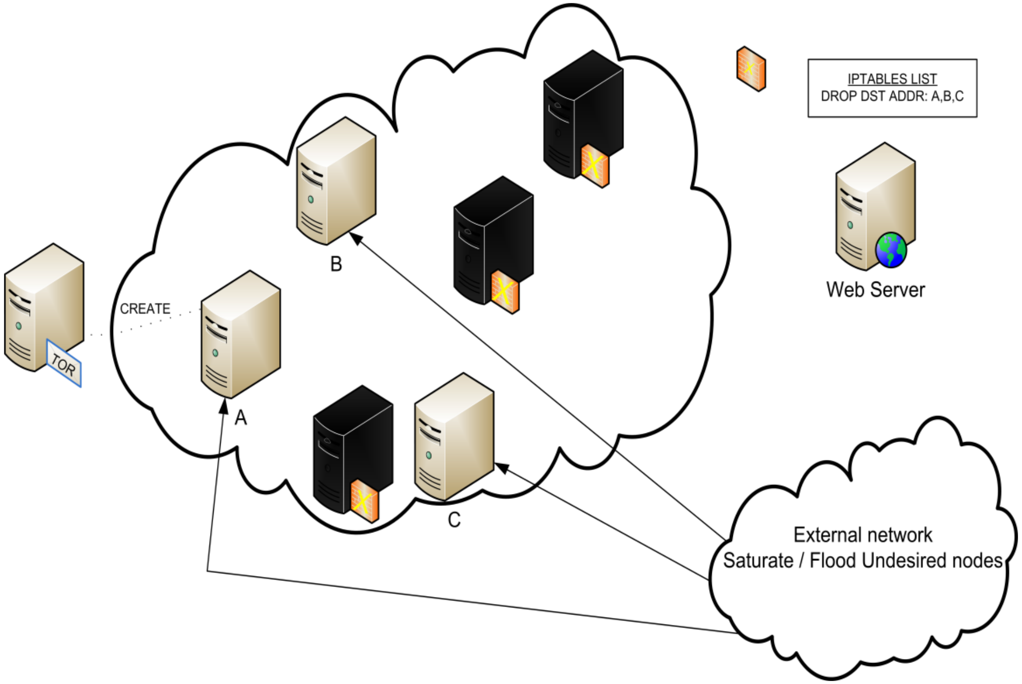
Attack Step 2: Denying a Subset of ORs

Force as most as possible of the 3-node routes to go through the compromised nodes (attack Conduct of manoeuvre (b))

- Selective DoS of a subset of ORs (public or non public) to force users (TOR clients) to go through the remaining ones with a very high probability.
- Inspired from existing works or from ISP own techniques, we have developed 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).

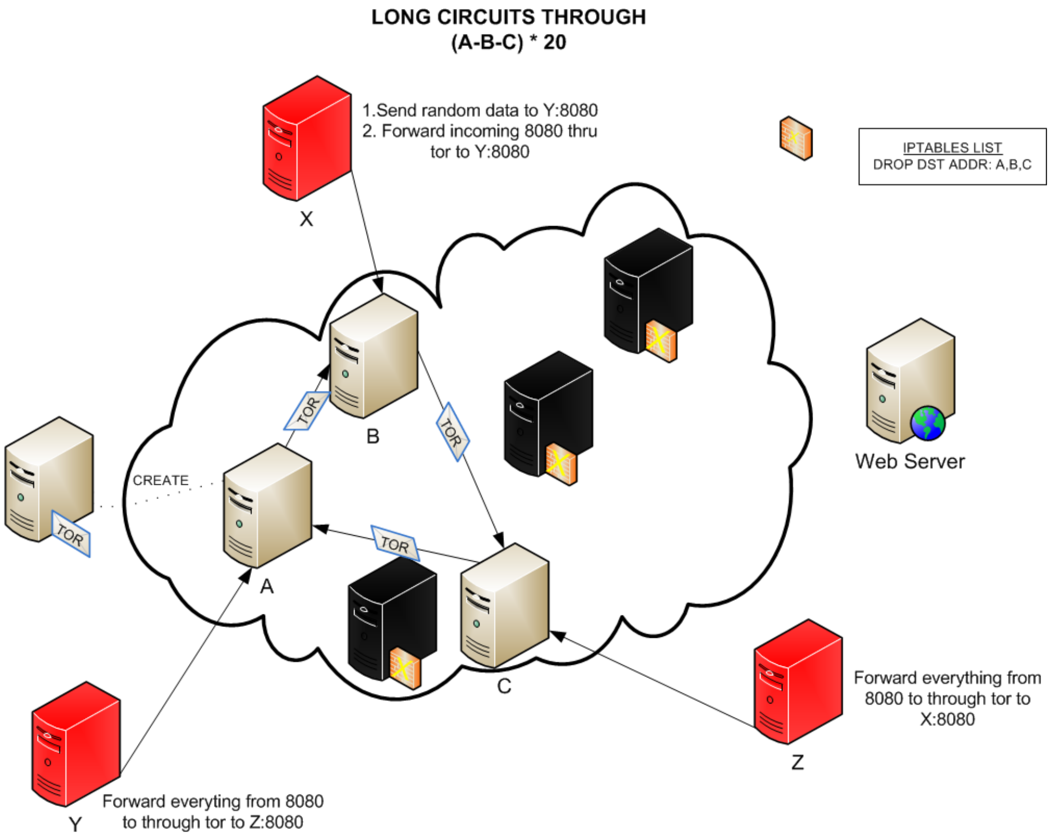


Attack Step 2: Congestion Attack and Path Selection (2)

- 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).

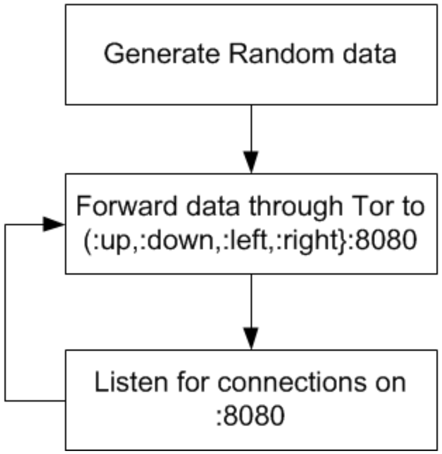
Attack Step 2: Packet spinning attack over long paths

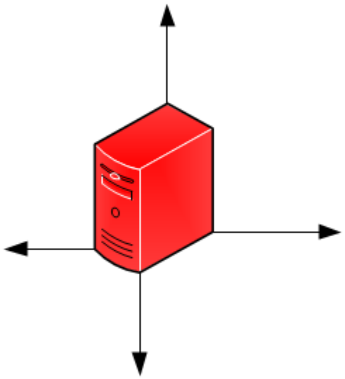
- Inspired from (Pappas et al., 2008).
- We block subsets of ORs (in white) by making them infinitely looping over long circuits (average 100 ORs). Quickly overload uncompromised nodes with high bandwidth
- This technique reduces the resources required to flood since the infrastructure is already in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


LONG CIRCUITS THROUGH
(A-B-C) * 20
1.Send random data to Y:8080
2. Forward incoming 8080 thru tor to Y:8080
X
IPTABLES LIST
DROP DST ADDR: A,B,C
B
TOR
TOR
CREATE
TOR
A
TOR
C
Web Server
Forward everything from 8080 to through tor to X:8080
Z
Forward everyting from 8080 to through tor to Z:8080
Y

Attack Step 2: Packet spinning attack over long paths

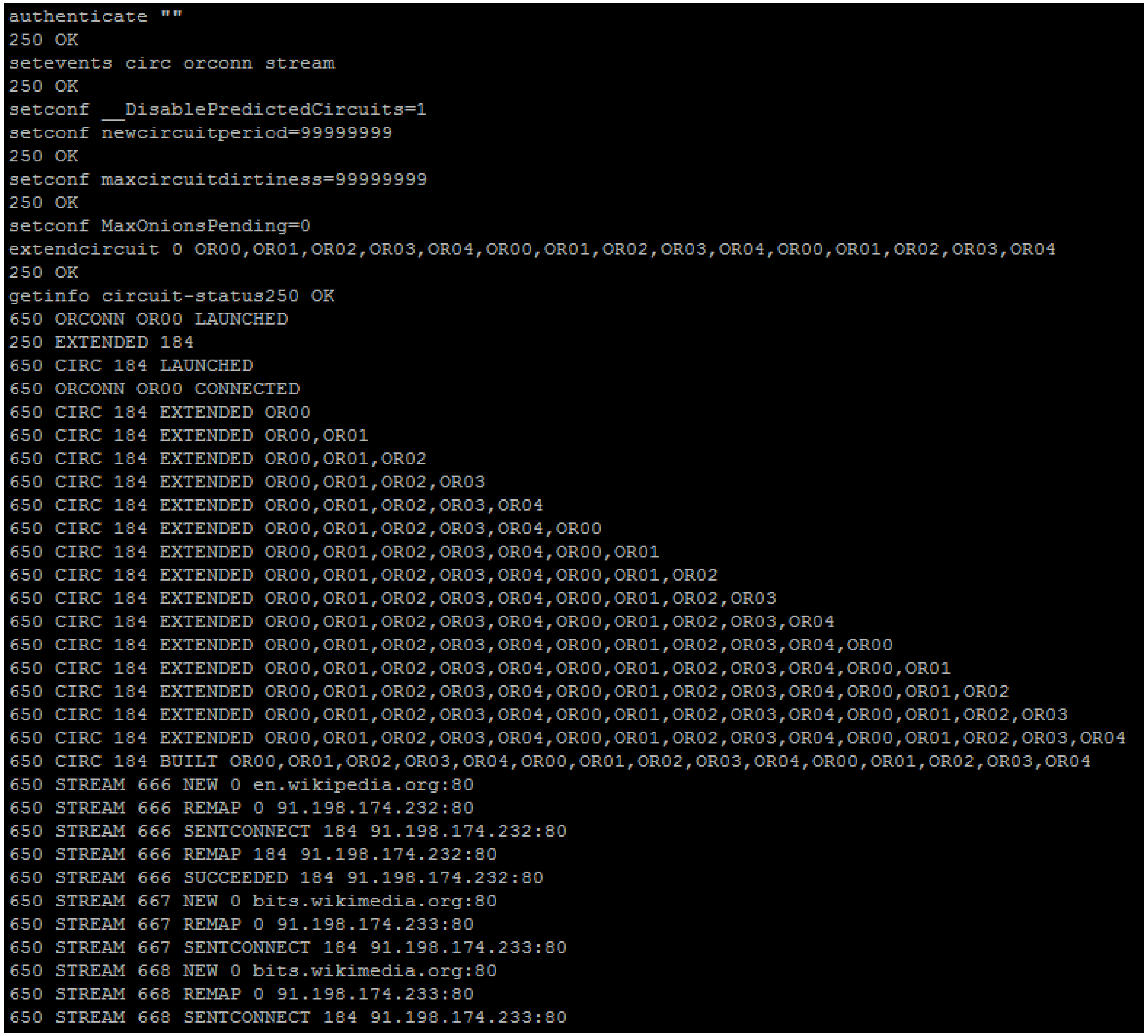
- Use the network itself to
- Flood itself with circular repetitive paths.
- Long paths to keep the packets flowing.
- Feedback into the circuit, maintain continuous and consistent overload.


Generate Random data
Forward data through Tor to (:up,:down,:left,:right}:8080
Listen for connections on :8080



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

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

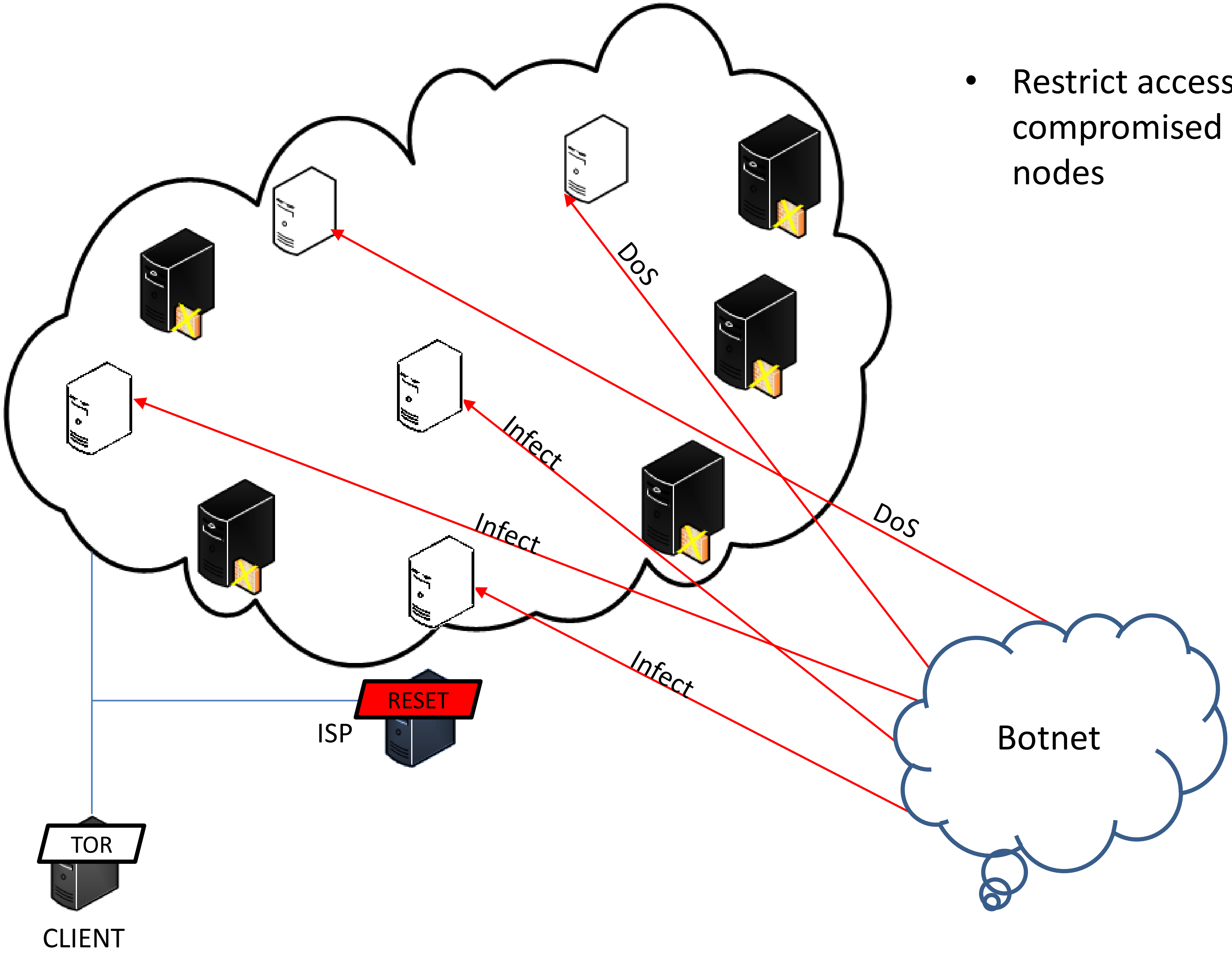


```
authenticate ""
250 OK
setevents circ orconn stream
250 OK
setconf __DisablePredictedCircuits=1
setconf newcircuitperiod=99999999
250 OK
setconf maxcircuitdirtiness=99999999
250 OK
setconf MaxOnionsPending=0
extendcircuit 0 OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04,OR00,OR01,OR02,OR03,OR04
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

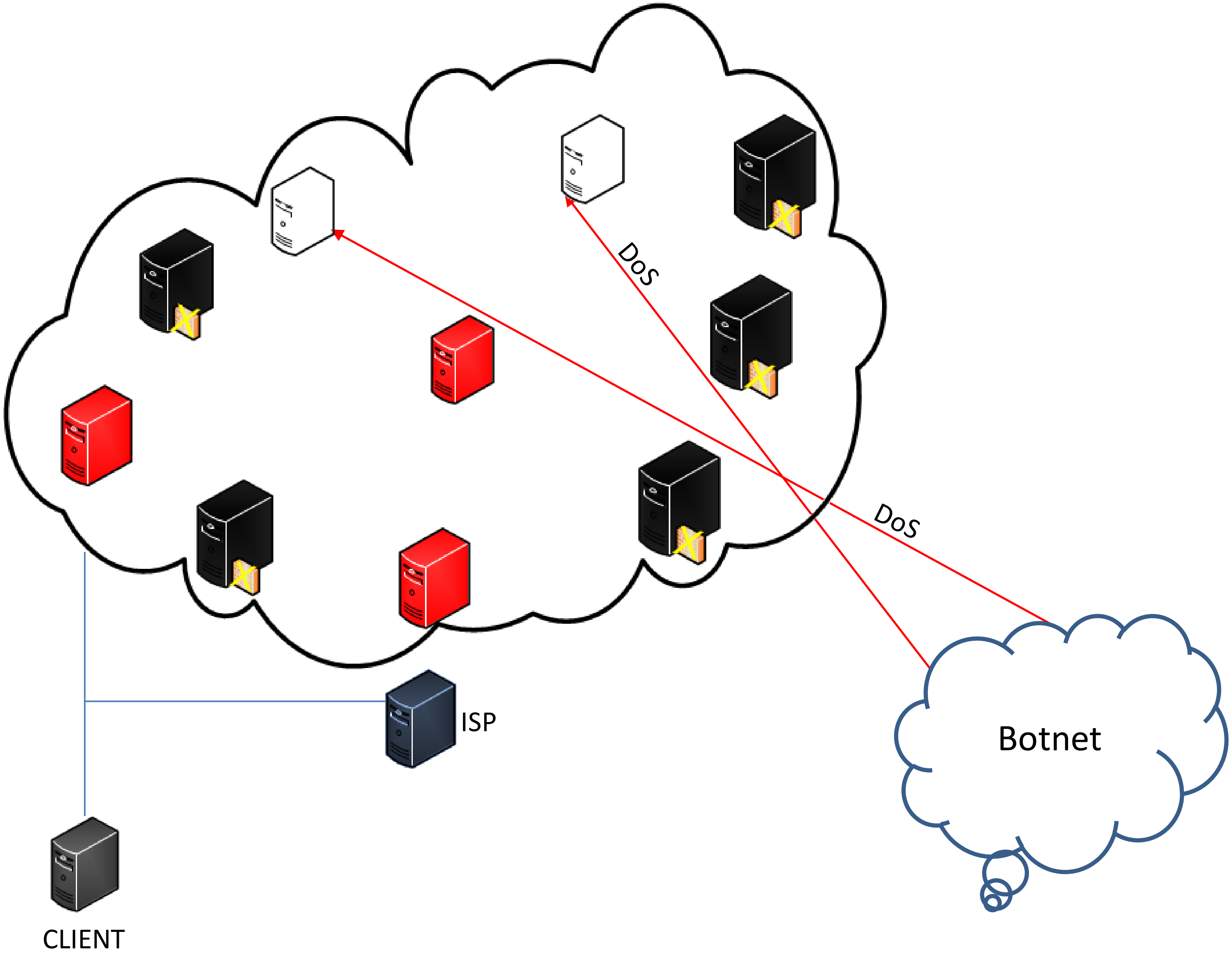
- We intend to force single clients to connect to a subset of (compromised) ORs.
- We assume that the attacker has access to the network of the client (case of ISPs).
  - This technique seems to be used by ISPs already.
- We mimic the behaviour of a typical ISP device by monitoring the packets and sending 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 that have a high metric (bandwidth and uptime).

Attack Step 2: All attacks



- Restrict access to only pre-compromised or controlled nodes

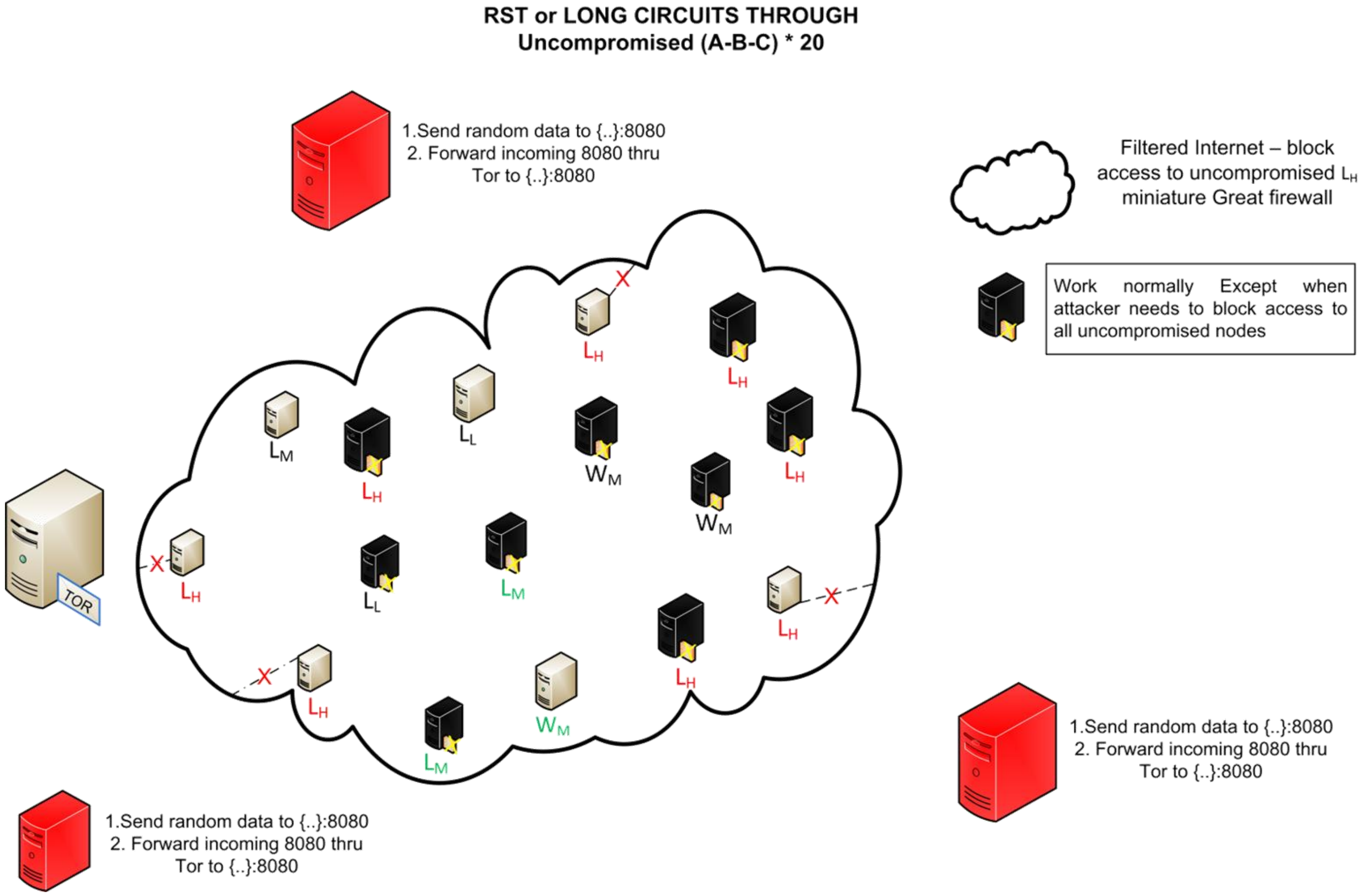
Attack Step 2: Desirable Onion cloud



Attack Step 2: TCP Reset Attack (2)

TCP RST attack scenario

- Restrict access to higher metric (bandwidth and uptime), uncompromised nodes



RST or LONG CIRCUITS THROUGH
Uncompromised (A-B-C) * 20

1.Send random data to {..}:8080
2. Forward incoming 8080 thru
Tor to {..}:8080

Filtered Internet – block access to uncompromised L_H miniature Great firewall

Work normally Except when attacker needs to block access to all uncompromised nodes

1.Send random data to {..}:8080
2. Forward incoming 8080 thru
Tor to {..}:8080

1.Send random data to {..}:8080
2. Forward incoming 8080 thru
Tor to {..}:8080

L => Linux
W => Windows

Subscripts denote high medium and low bandwidths.

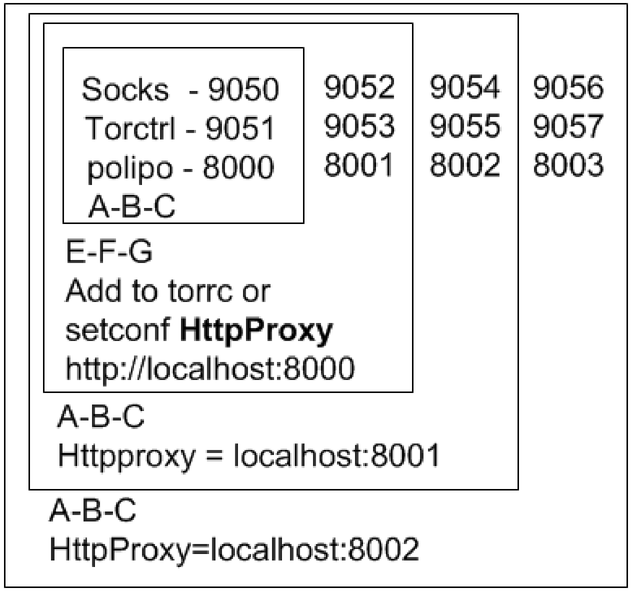
Attack Step 2: TCP Reset Attack (2)

Tor_extend-2.0.0

- In the event that proposal 110 (https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/110-avoid-infinite-circuits.txt) is enabled fully, it will still be possible to establish long circuits in this way.
- Suitable exit nodes must be chosen with no restrictions for the entry nodes used

Considered in (Dingledine et. al, 2009)

orarray = A,B,C,E,F,G,A,B,C,A,B,C



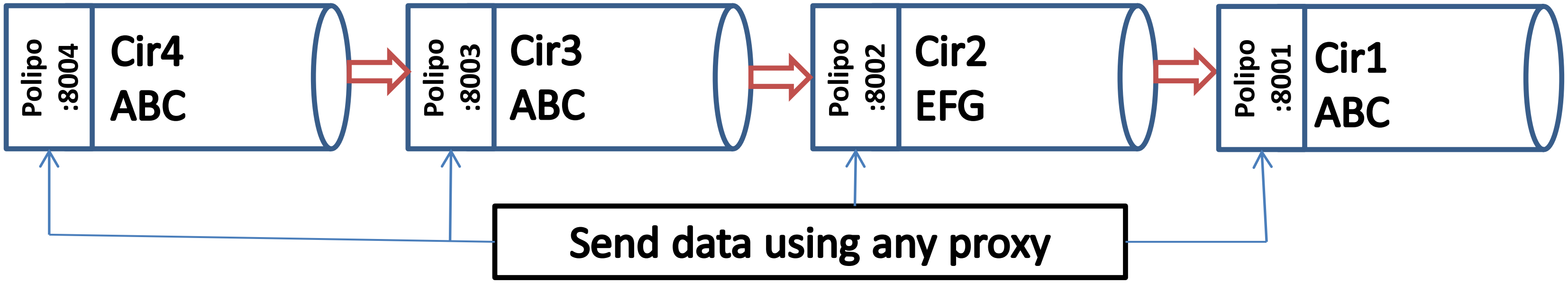
Socks - 9050
Torctrl - 9051
polipo - 8000
A-B-C

E-F-G
Add to torrc or
setconf **HttpProxy** http://localhost:8000

A-B-C
Httpproxy = localhost:8001

A-B-C
HttpProxy=localhost:8002

9052	9054	9056
9053	9055	9057
8001	8002	8003



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 layer combines 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 (time window) are “*parallel cipher texts*”. Use *Medigo* 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/deactivation 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 negotiation 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 traffic is a success already!

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 all COMSEC only-based solutions).
- 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 tools 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

1. Introduction
2. Dynamic cryptographic trapdoors
   - Introduction
   - OS level dynamic trapdoors
   - Algorithm level dynamic trapdoor
3. Taking Over the Tor network
   - Tor network description
   - Cryptography and security in Tor network
   - Taking control over the Tor network
4. Conclusion

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.
- Do not underestimate the international police forces coordination capabilities with ISPs!
- 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.

Conclusion: Tor Changes…

Draft of Future Crypto Proposals from Tor Development (November 2011) (https://lists.torproject.org/pipermail/tor-dev/2011-November/002999.html)
"FOR A STREAM CIPHER: AES-CTR is in one sense a conservative choice inasmuch as AES is well-analyzed, but AES's **well-known issues** with cache-based timing attacks are pretty worrisome. We can mitigate some by **using random secret IVs for AES-CTR**, so that we will be encrypting neither attacker-chosen nor attacker-known plaintext with our AES cipher, but that's a bit kludgy. There are also supposed to be time-invariant implementations that use Intel's AESNI instructions where available, and time-invariant implementations that use bit-slicing."

If the malware influenced the IV as in part 1 of this talk, it's just a matter time!

- The operating systems should be protected from such high level memory operations
- Crypto API's should include protections against such dynamic trapdoors.
- Applications developers, in this case Tor-dev team, should do what they can to prevent this too
- Algorithms might be standardised, but implementations can be flawed
- Murdoch (2011), "Comparison of Datagram", can serve as a starting point to replace TCP, thus preventing some of the TCP drawbacks and congestion. https://blog.torproject.org/blog/moving-tor-datagram-transport

Conclusion: Tor Updates since H2HC and PacSec

November

- Unconditionally use OpenSSL's AES implementation instead of our old built-in one. OpenSSL's AES has been better for a while, and relatively few servers should still be on any version of OpenSSL that doesn't have good optimized assembly AES.
  - Use OpenSSL's EVP interface for AES encryption, so that all AES operations can use hardware acceleration (if present). Resolves ticket 4442. (Crypto **Performance**)

December

- When using OpenSSL 1.0.0 or later, use OpenSSL's counter mode implementation. It makes AES_CTR about 7% faster than our old one (which was about 10% faster than the one OpenSSL used to provide). Resolves ticket 4526. (Crypto **Performance**)
- Only use the EVP interface when AES acceleration is enabled, to avoid a 5-7% performance regression. Resolves issue 4525; bugfix on 0.2.3.8-alpha. (Crypto **Performance**)
- Make bridge SSL certificates a bit more stealthy by using random serial numbers, in the same fashion as OpenSSL when generating self-signed certificates. Implements ticket 4584. (Randomness)
- Introduce a new config option "DynamicDHGroups", enabled by default, which provides each bridge with a unique prime DH modulus to be used during SSL handshakes. This option attempts to help against censors who might use the Apache DH modulus as a static identifier for bridges. Addresses ticket 4548.

The END

Many thanks for your attention.
Everything on http://cvo-lab.blogspot.com/2011/11/tor-attack-technical-details.html

Questions and answers!