SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
Trustworthy ICT

NETWORK OF EXCELLENCE

A European Network of Excellence in Managing Threats and Vulnerabilities in the Future Internet: *Europe for the World* †

Deliverable D2.4: 2nd Project Workshop Proceedings

Abstract: This document contains the pre-proceedings of the *SysSec* 2nd Project Workshop, which took place in Amsterdam on July the 6th, co-located with the UbiCrypt 2013 Summer School (RUB).

Responsible Partner	**Politecnico di Milano**
Contractual Date of Delivery	August 2013
Actual Date of Delivery	September 2013
Deliverable Dissemination Level	Public
Editor	Federico Maggi
QMC Reviewers	Evangelos Markatos, Ali Rezaki

The *SysSec* consortium consists of:

FORTH-ICS	Coordinator	Greece
Politecnico Di Milano	Principal Contractor	Italy
Vrije Universiteit Amsterdam	Principal Contractor	The Netherlands
Institut Eurécom	Principal Contractor	France
IICT-BAS	Principal Contractor	Bulgaria
Technical University of Vienna	Principal Contractor	Austria
Chalmers University	Principal Contractor	Sweden
TUBITAK-BILGEM	Principal Contractor	Turkey

† The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007.

Contents

1 Introduction and Organization

1.1 Introduction to the Event

The Second Project Workshop aimed to consolidate the Systems Security research community in Europe. The specific format of this workshop has been developed to:

- showcase and spread the excellence in systems security research in Europe, by presenting a selection of papers published by European researchers and Europe-funded research projects in top conferences in the area;
- involve students and young researchers by allowing them to showcase their own best results and expose them to top researchers in the field;
- create a generational exchange between experienced and starting researchers, focusing around a tutorial on how to get your research published in top venues (a session discussing the ”best previously rejected papers” of the last years). For this reason, we decided to co-locate the workshop with the UbiCrypt Summer School 2013.

While the First Project Workshop aimed at mapping the research of the systems security groups in EU, the Second Project Workshop aimed at showing and disseminating the top results from those groups.

The resulting program was well received by all the participating students, who often interacted with the speakers both during and after the talks.

Bochum, 24 July 2013

Stefano Zanero, General Chair.

1.2 Committees and Organization

The workshop was co-located with the UbiCrypt Summer School 2013 on Reverse Engineering, which took place from July 22nd to July 26th. The school offered graduate students and young researchers the opportunity to learn more about binary analysis and malware reverse engineering.

Poster Session Programme Committee

Davide Balzarotti, Institut Eurecom

Herbert Bos, Vrije Universiteit Amsterdam

Thorsten Holz, Ruhr University Bochum

Federico Maggi, Politecnico di Milano

Stefano Zanero, Politecnico di Milano

Publicity Chair and Proceedings Editor

Federico Maggi, Politecnico di Milano

Local Organization Chair

Thorsten Holz, Ruhr University Bochum, Germany

2 Presentations

This chapter contains copies of the slides used by the speakers for their workshop presentations.

It should be noted that we asked all of the speakers to flavor their presentation so that it would teach students how to write a great paper for a top-tier technical conference, and what type of excellence in research is spread around in the systems community in Europe.

To achieve these objectives, we structured the workshop in three sessions. In Session 2.1 papers from top-tier conferences by top EU researchers were presented. This would give students a glimpse of research excellence and what it means. In Session 2.2 two colleagues graciously accepted to talk about their best rejects: papers that were rejected before being accepted in a top conference. They presented this as a collection of lessons learned on how to get a paper published in a highly rated venue. Finally, in Session 2.3 we showcased the contribution of the European Commission and the Seventh Framework Programme, by hosting and showcasing excellent research by EU-funded projects.

2.1 Session 1: Top Papers From Europe

In this session we invited the presentation of papers from top-tier conferences, to expose the students to the excellence in research represented by some of the top EU researchers in the systems security field.

2.1.1 Prudent Practices for Designing Malware Experiments: Status Quo and Outlook

Authors Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, Maarten van Steen.

Speaker Christian Rossow.

Paper summary Malware researchers rely on the observation of malicious code in execution to collect datasets for a wide array of experiments, including generation of detection models, study of longitudinal behavior, and validation of prior research. For such research to reflect prudent science, the work needs to address a number of concerns relating to the correct and representative use of the datasets, presentation of methodology in a fashion sufficiently transparent to enable reproducibility, and due consideration of the need not to harm others. In this paper we study the methodological rigor and prudence in 36 academic publications from 2006-2011 that rely on malware execution. 40% of these papers appeared in the 6 highest-ranked academic security conferences. We find frequent shortcomings, including problematic assumptions regarding the use of execution-driven datasets (25% of the papers), absence of description of security precautions taken during experiments (71% of the articles), and oftentimes insufficient description of the experimental setup. Deficiencies occur in top-tier venues and elsewhere alike, highlighting a need for the community to improve its handling of malware datasets. In the hope of aiding authors, reviewers, and readers, we frame guidelines regarding transparency, realism, correctness, and safety for collecting and using malware datasets.

Prudent Practices for Designing Malware Experiments

Do's and Dont's for Your Future Academic Career

UbiCrypt Summer School, July 2013- Christian Rossow

Malware Experiments

- Security researchers deploy experiments to
  - … **analyze** malware
  - … **cluster** malware
  - … **detect** malware
  - … **monitor** malware
  - … **infiltrate** malware

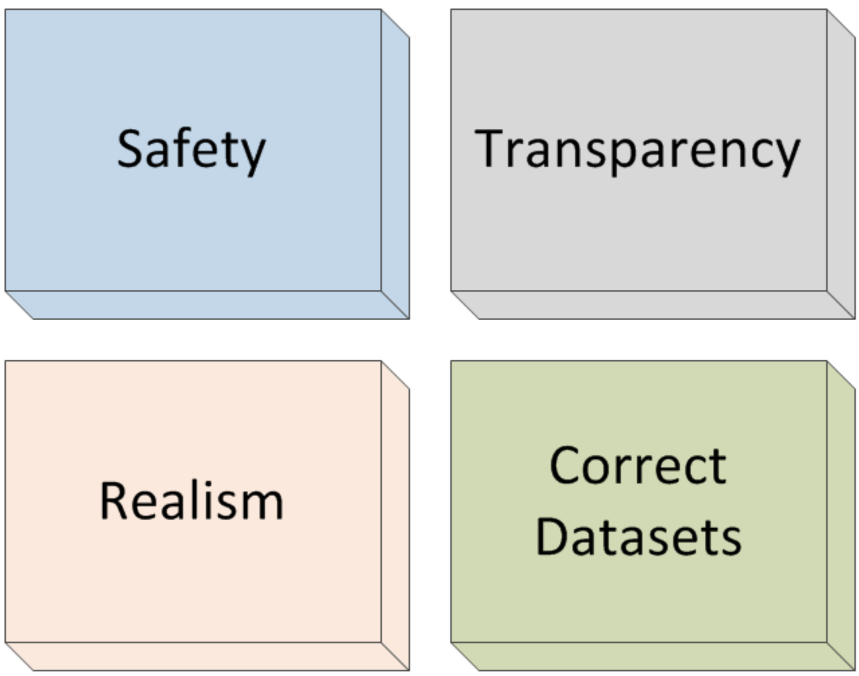
- Doing malware research is challenging

Running Example

- Alice aims to detect network traffic of malware

- Alice's plan:
  a. Dynamically analyze malware
  b. Record malware's network traffic
  c. Train a classifier based on traffic analysis
  d. Evaluate classifier on lab traffic

Guidelines for Prudent Malware Experiments

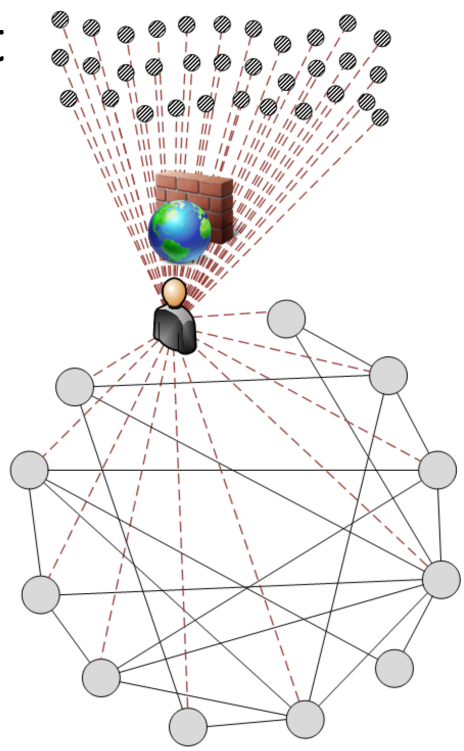


Guidelines: Safety

- Deploy containment policies
  - Malware causes harm to others
  - Redirect attacks (spam, DDoS) to local targets
  - Throttle amount of traffic
- Describe your policies
  - Policies will influence your results
  - Discuss your decisions

Guidelines: Transparency

- Describe execution environment
  - Which OS / software configuration?
  - Which network connectivity? NAT?



Guidelines: Transparency

- Analyze the reasons for FPs/FNs
  - When did it succeed? Why did it fail?
  - How can it be optimized / circumvented?

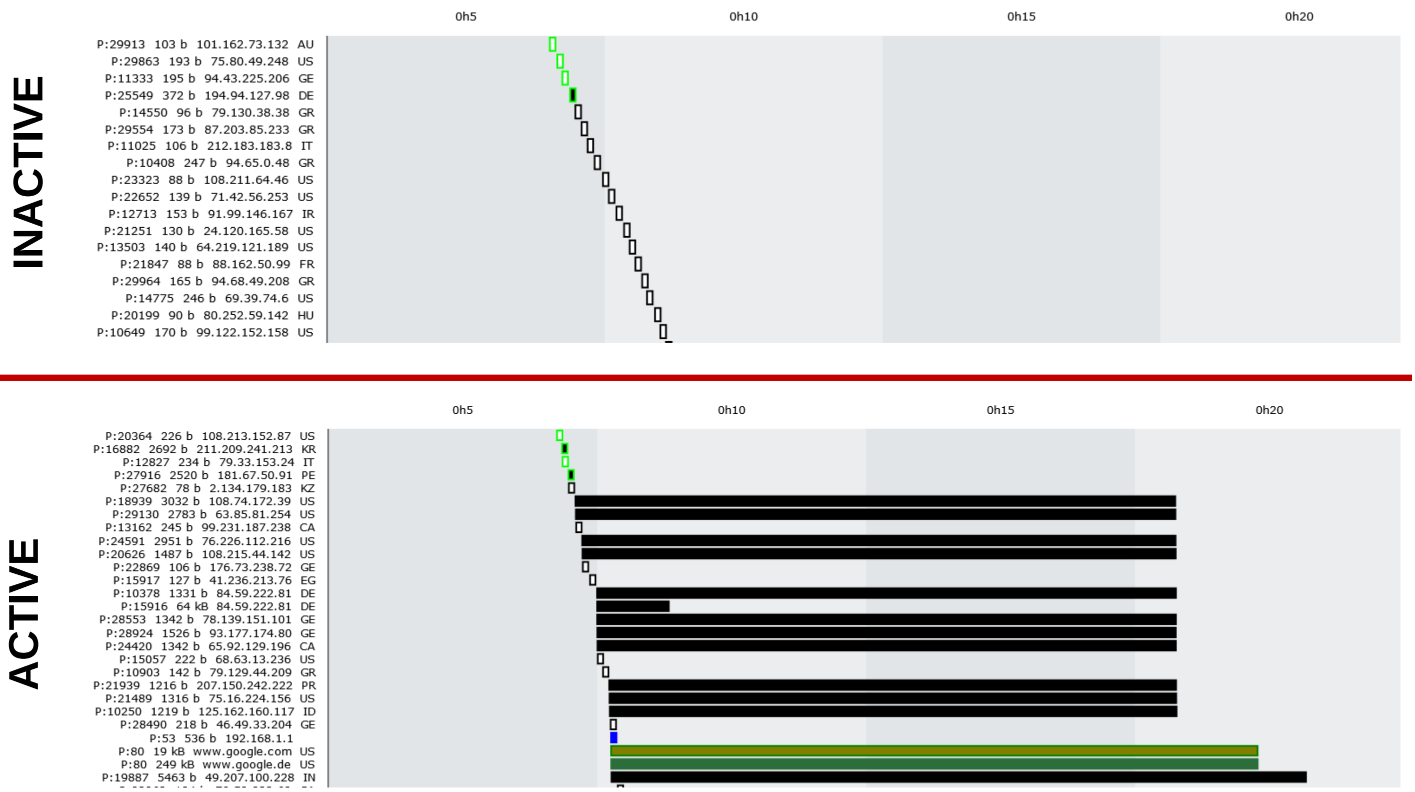
"We have a super-low and stunning 0.05% False Positive Rate." → "We have observed three FPs because of X and Y. We could (not) counter these FPs by …"

"We detected all bots with a False Negative Rate of 0%." → "We detected the C&C flow of all bots. The C&C flows were characteristic because X and Y…"

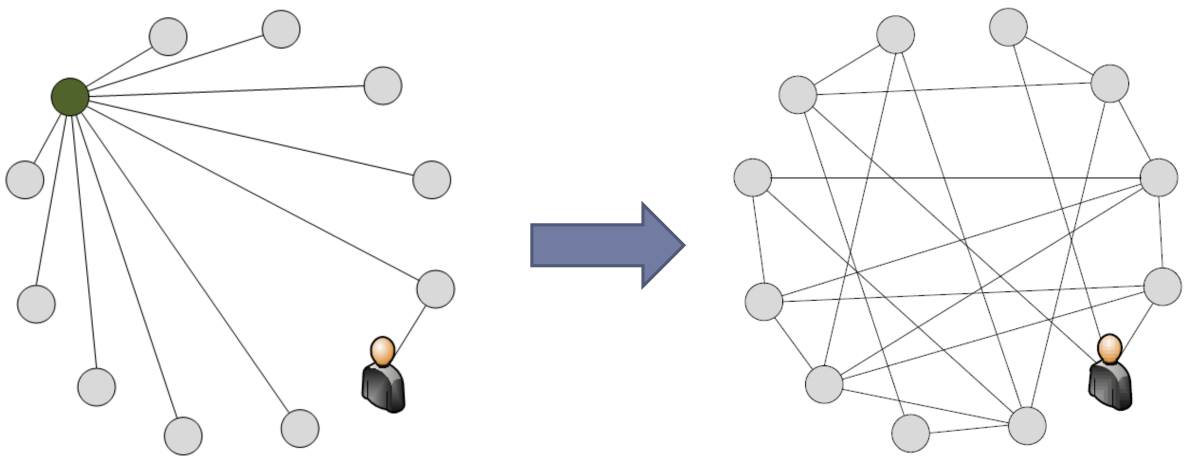
Guidelines: Realism

- Evaluate relevant malware families
  - Do not analyze years-old malware samples
  - Focus on popular and recent malware
  - Give thought to sufficient sampling sizes
- Detect malware in real-world scenarios
  - On live traffic and with real users
  - Otherwise you may get artificial results

Guidelines: Correct Datasets

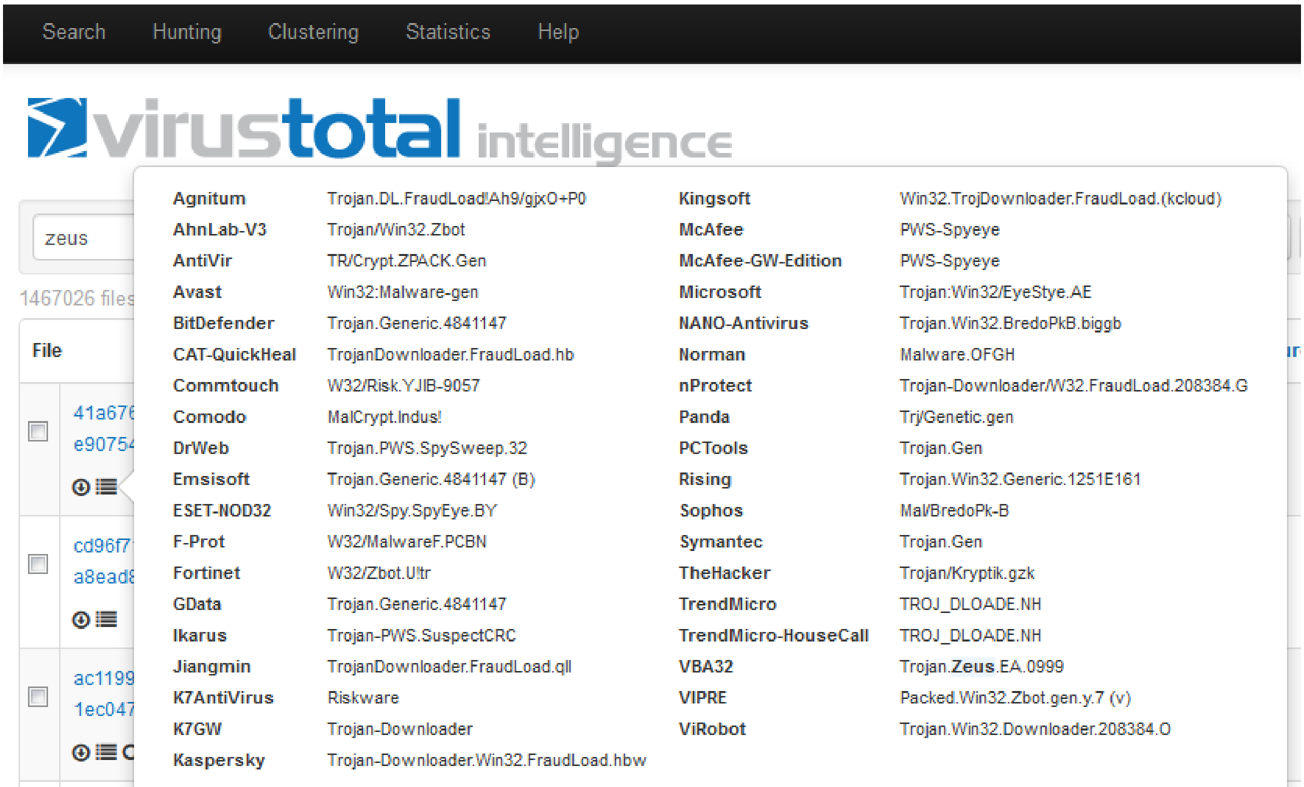


Excursion: Zeus P2P Sinkholing

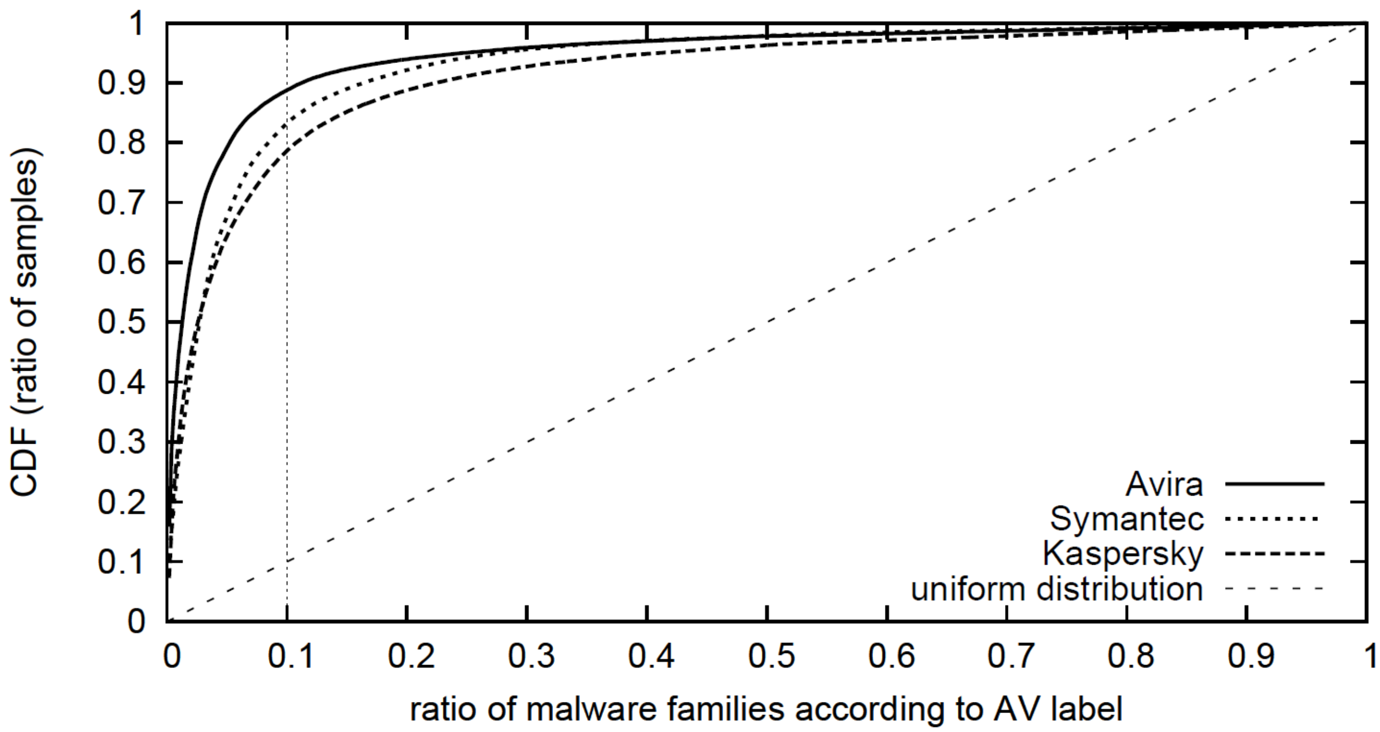


(Details see “P2PWNED” @ IEEE S&P 2013)

Guidelines: Correct Datasets

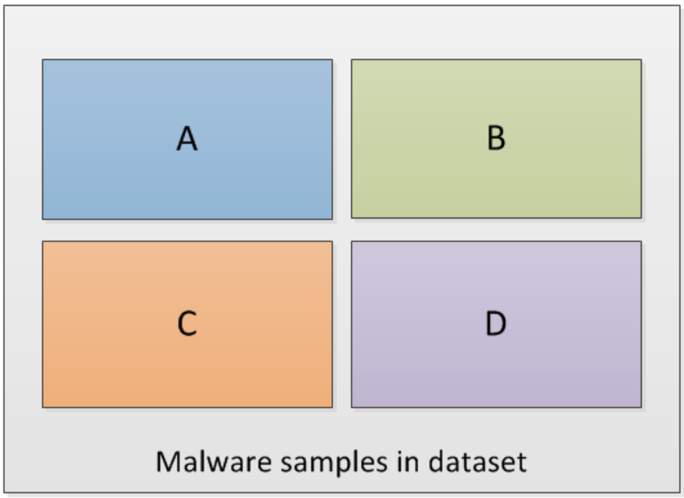


Guidelines: Correct Datasets



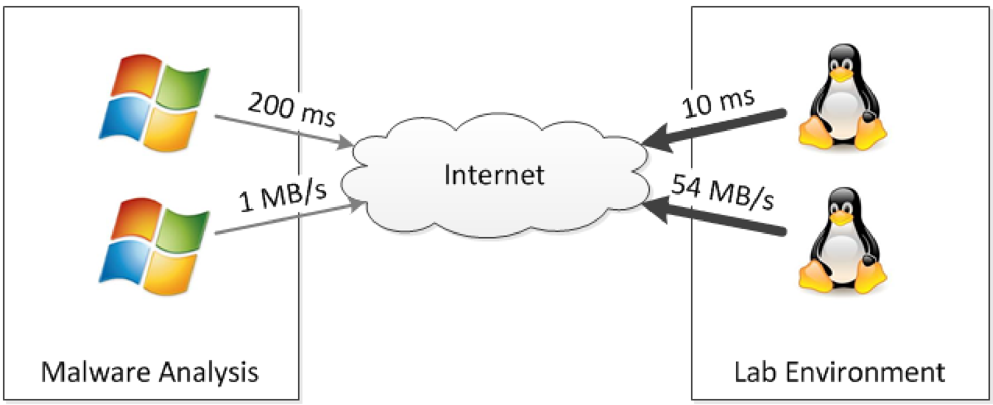
Guidelines: Correct Datasets

- Balance datasets over malware families
  - Malware polymorphism can skew distributions
  - This in turn skews the evaluation
  - “We detect 90%” (… so only 1 family?)



Guidelines: Correct Datasets

- Be aware of artifacts
  - Specific artifacts in contained environments
  - Use caution when blending malware activity traces into benign background activity



Conclusion for Alice

a. Dynamically analyze malware

- Make sure it's malware (and let it be active)
- Balance according to malware families
- Containment policies!

b. Record malware's network traffic

- Avoid sinkholes
- What traffic? All? Only C&C?!

c. Train a classifier based on traffic analysis

- Remove artifacts
- Behavior depends on environment config

d. Evaluate classifier on lab traffic

- Be realistic – real world!

Lessons Learned

- Choose a specific target
  - Bad: I want to detect malware
  - Good: I want to detect crypted C&C communication
- Evaluate carefully and thoroughly
  - Know your datasets
  - Interpret your results
  - Analyze strengths/weaknesses

Prudent Practices for Designing Malware Experiments

@christianrossow

Thanks to my co-authors: C. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, M. van Steen

2.1.2 Before We Knew It

Authors Leyla Bilge, Tudor Dumitras.

Speaker Leyla Bilge.

Paper Summary Little is known about the duration and prevalence of zero-day attacks, which exploit vulnerabilities that have not been disclosed publicly. Knowledge of new vulnerabilities gives cyber criminals a free pass to attack any target of their choosing, while remaining undetected. Unfortunately, these serious threats are difficult to analyze, because, in general, data is not available until after an attack is discovered. Moreover, zero-day attacks are rare events that are unlikely to be observed in honeypots or in lab experiments. In this paper, we describe a method for automatically identifying zero-day attacks from field-gathered data that records when benign and malicious binaries are downloaded on 11 million real hosts around the world. Searching this data set for malicious files that exploit known vulnerabilities indicates which files appeared on the Internet before the corresponding vulnerabilities were disclosed. We identify 18 vulnerabilities exploited before disclosure, of which 11 were not previously known to have been employed in zero-day attacks. We also find that a typical zero-day attack lasts 312 days on average and that, after vulnerabilities are disclosed publicly, the volume of attacks exploiting them increases by up to 5 orders of magnitude.

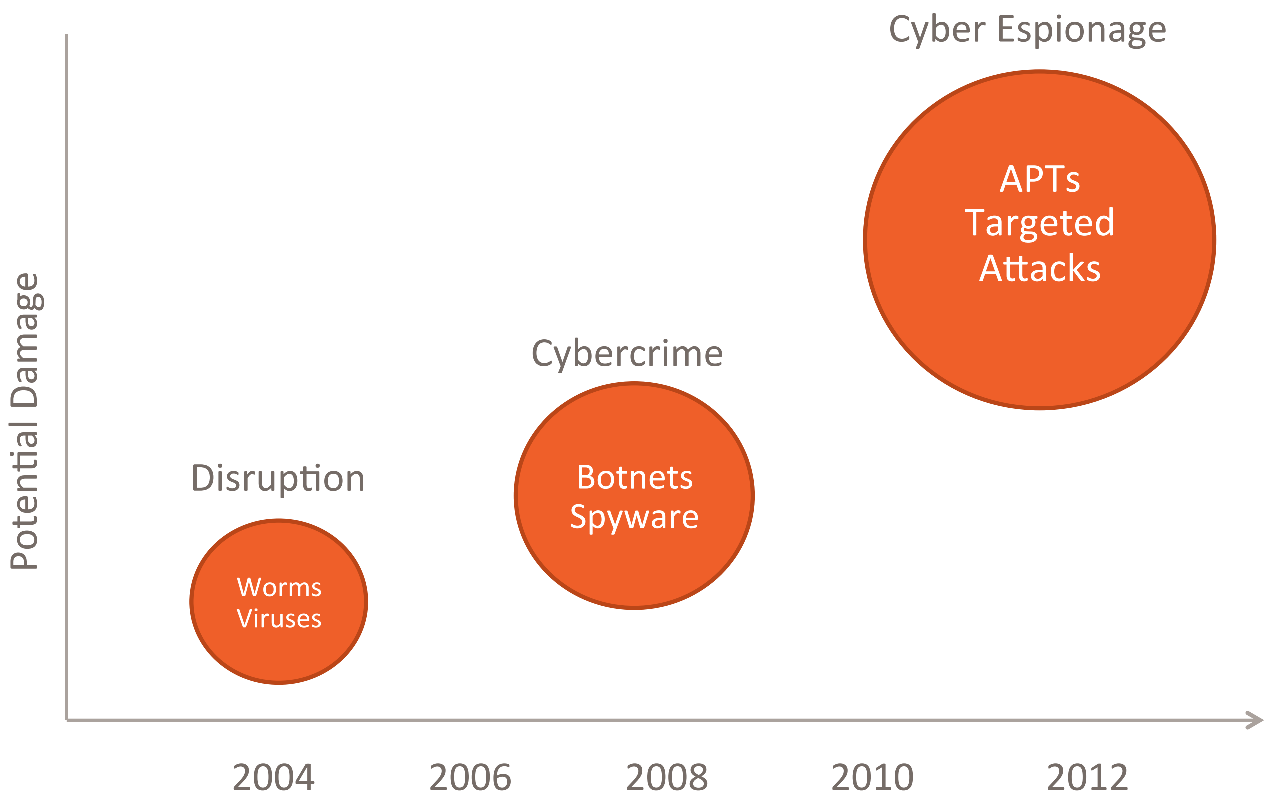
Before We Knew It

An Empirical Study of Zero-Day Attacks in the Real World

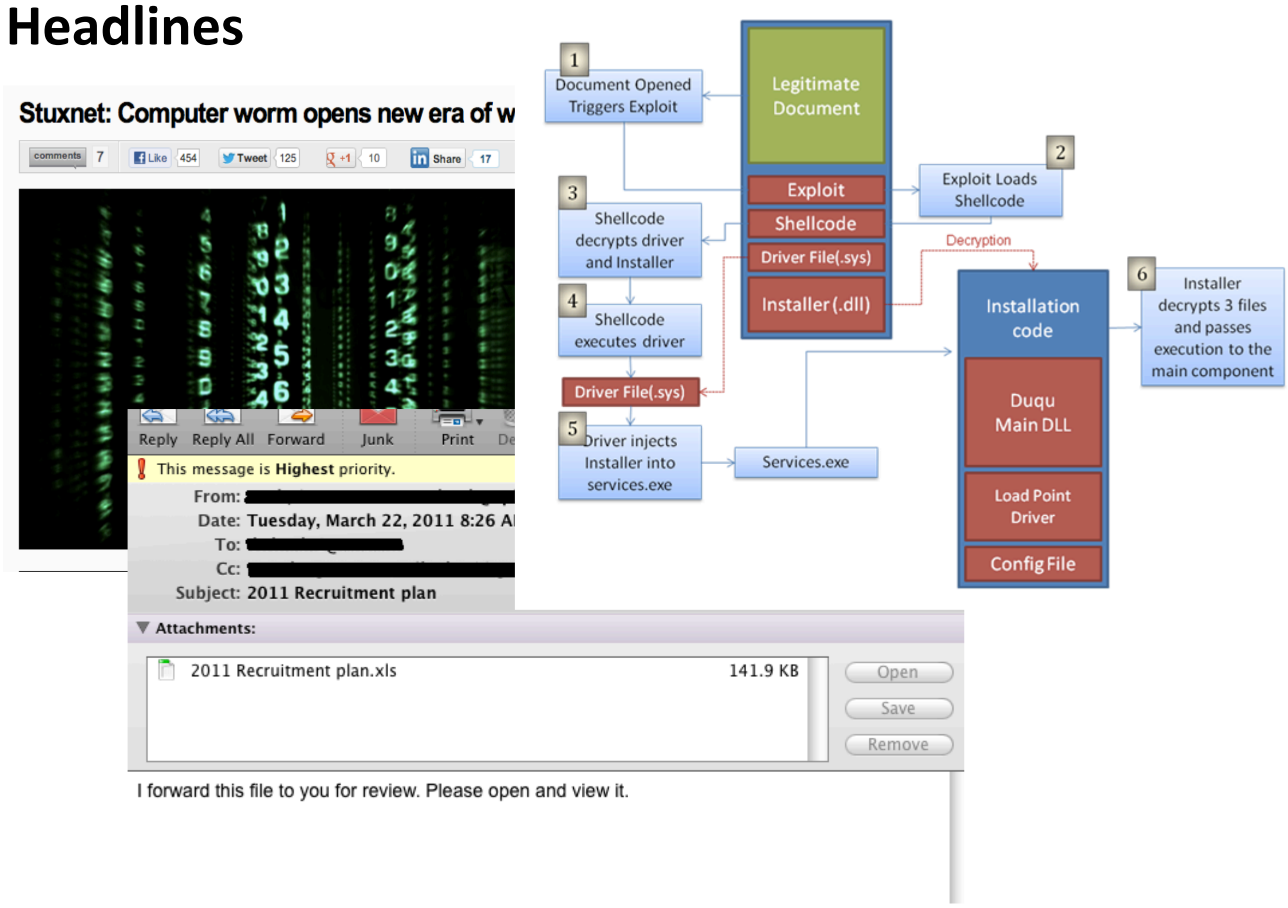
Leyla Bilge and Tudor Dumitraș

Symantec Research Labs

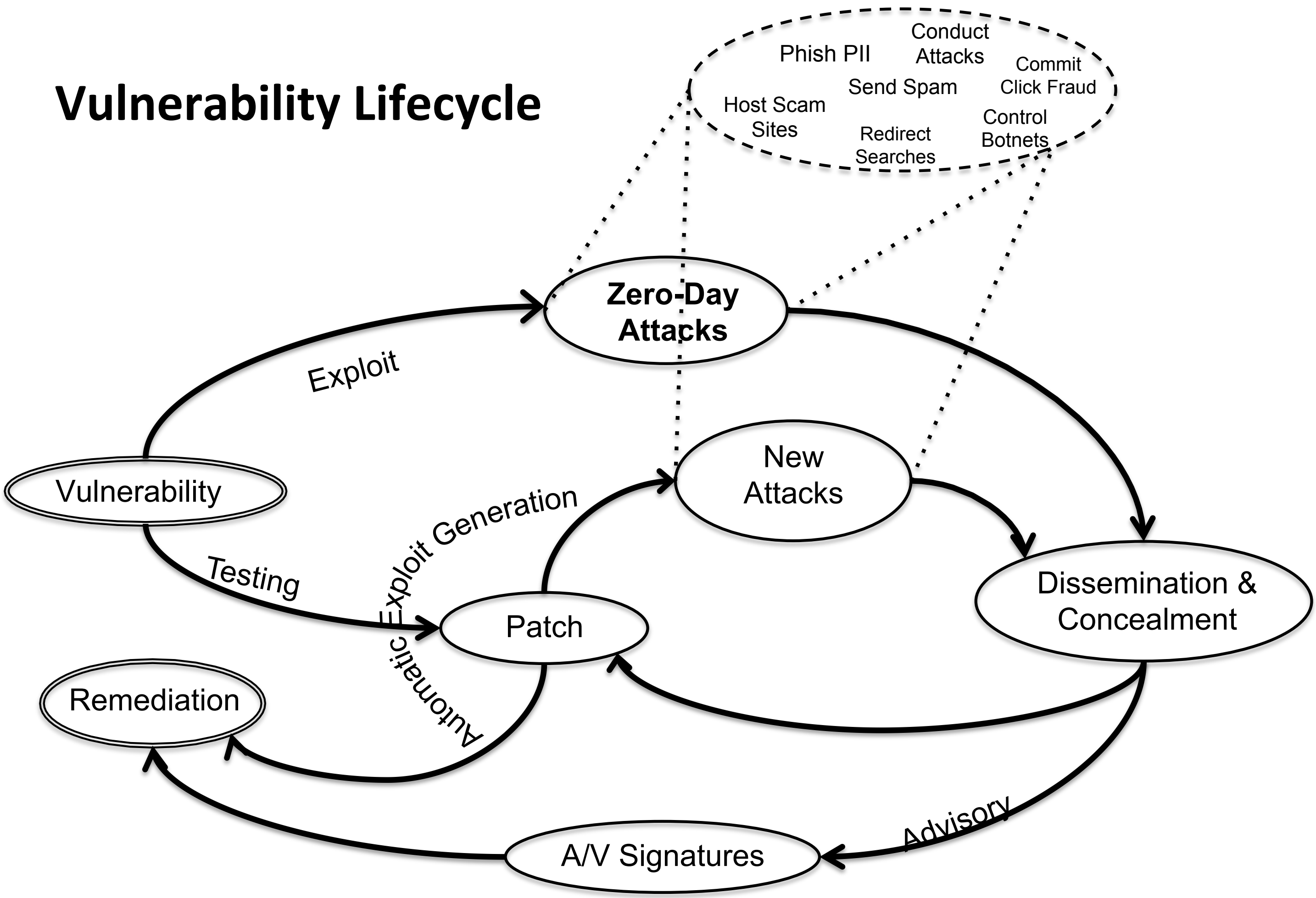
Threat Evolution



Headlines



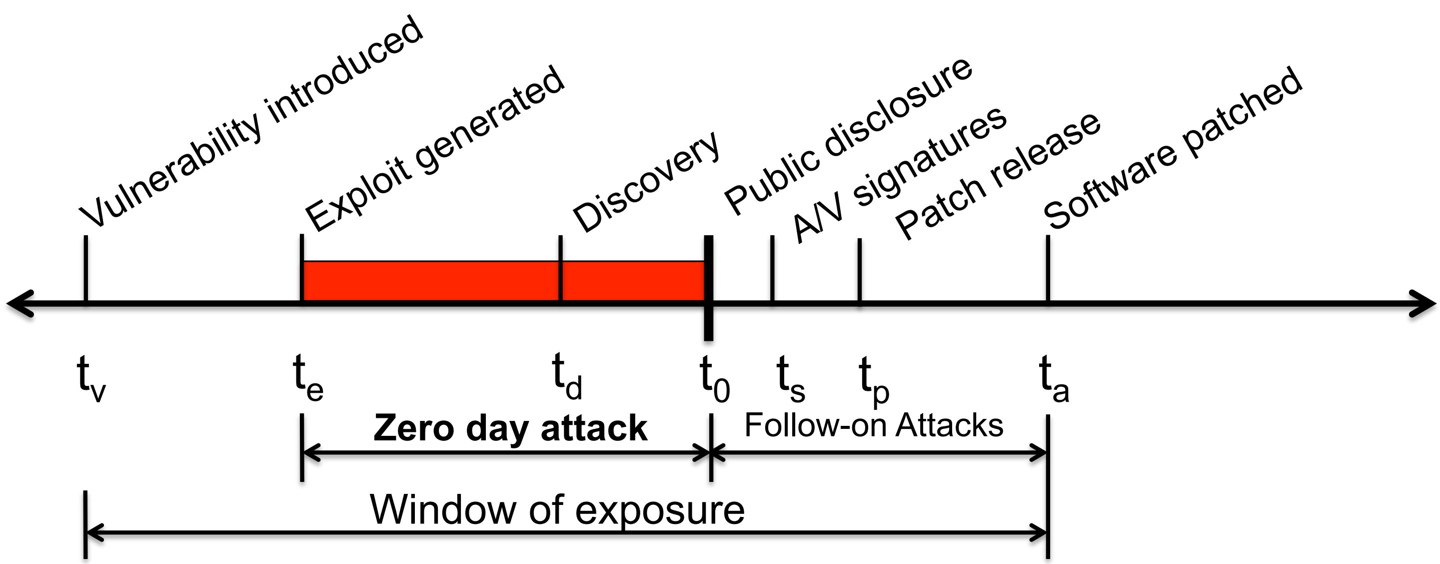
Vulnerability Lifecycle



Zero-day (0-day, Day zero) Attacks

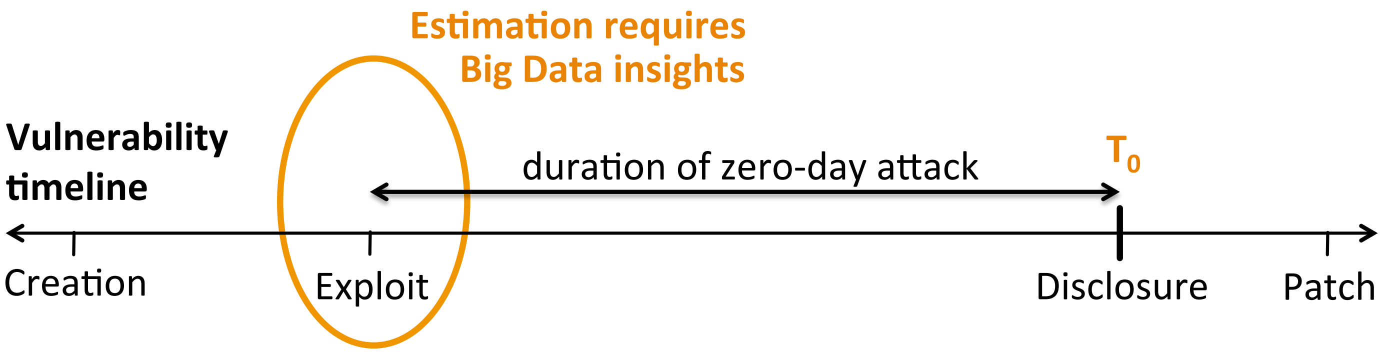
- Takes advantage of unknown vulnerabilities on programs before
  - They are discovered
  - They are publicly disclosed
  - A security patch is provided by the software vendor
- Common definition
  - An attack that uses a zero-day (0-day) exploit

0-day attacks and window of exposure



Research Questions

- **Are there more** zero-day vulnerabilities in the wild that we are not aware of?
- What is the typical **duration of zero-day attacks**?
- What is the **prevalence** of zero-day attacks?



WINE

Worldwide Intelligence Network Environment

Global Intelligence Network

Identifies more threats, takes action faster & prevents impact

Worldwide Coverage — **Global Scope and Scale** — *24x7 Event Logging*

Rapid Detection

Attack Activity
- 65M sensors
- 200+ countries

Malware Intelligence
- 133M client, server, gateways monitored
- Global coverage

Vulnerabilities
- 40,000+ vulnerabilities
- 14,000 vendors
- 105,000 technologies

Spam/Phishing
- 5M decoy accounts
- 8B+ email messages/day
- 1B+ web requests/day

Preemptive Security Alerts — **Information Protection** — *Threat Triggered Actions*

WINE Datasets

A/V telemetry:
136M machines

Malware:
7M samples

URL reputation:
10M domains

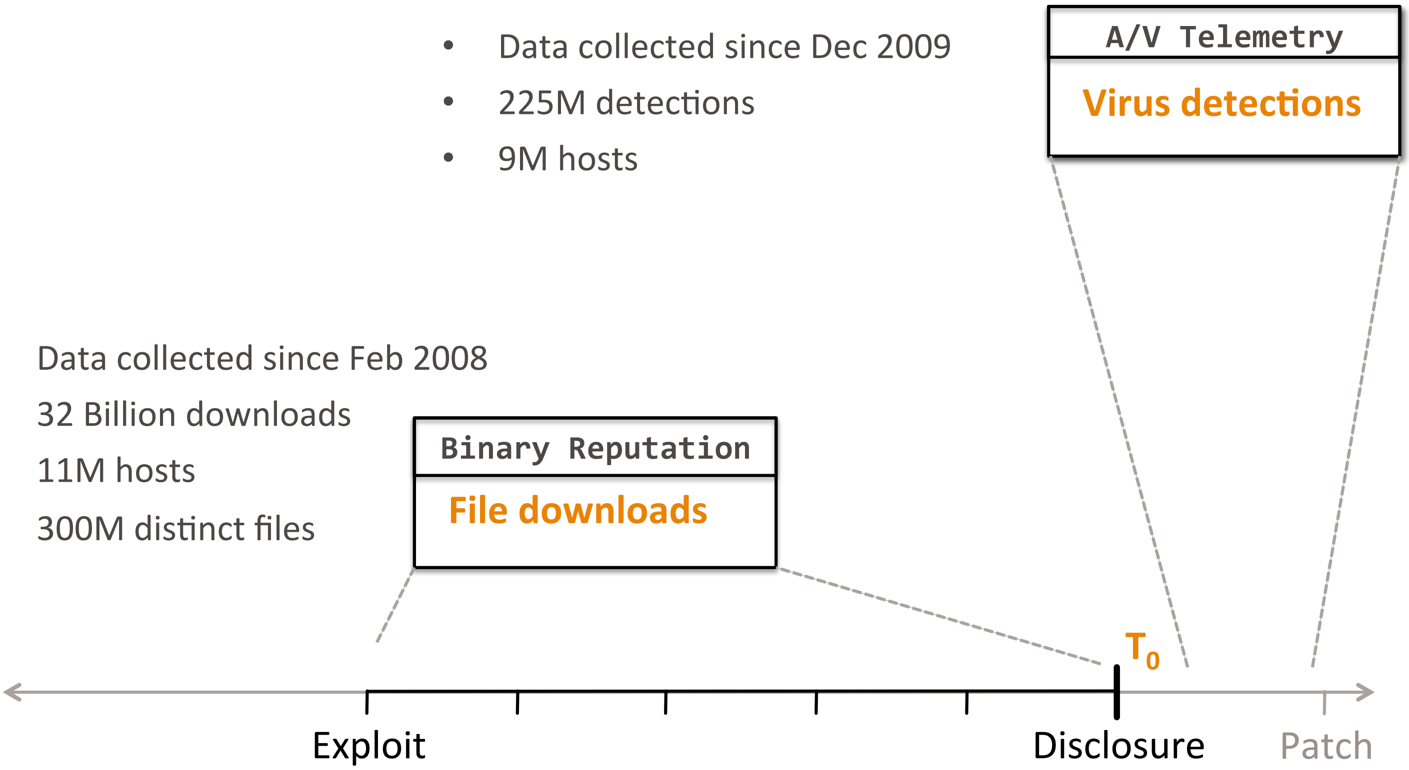
Binary reputation:
35M machines

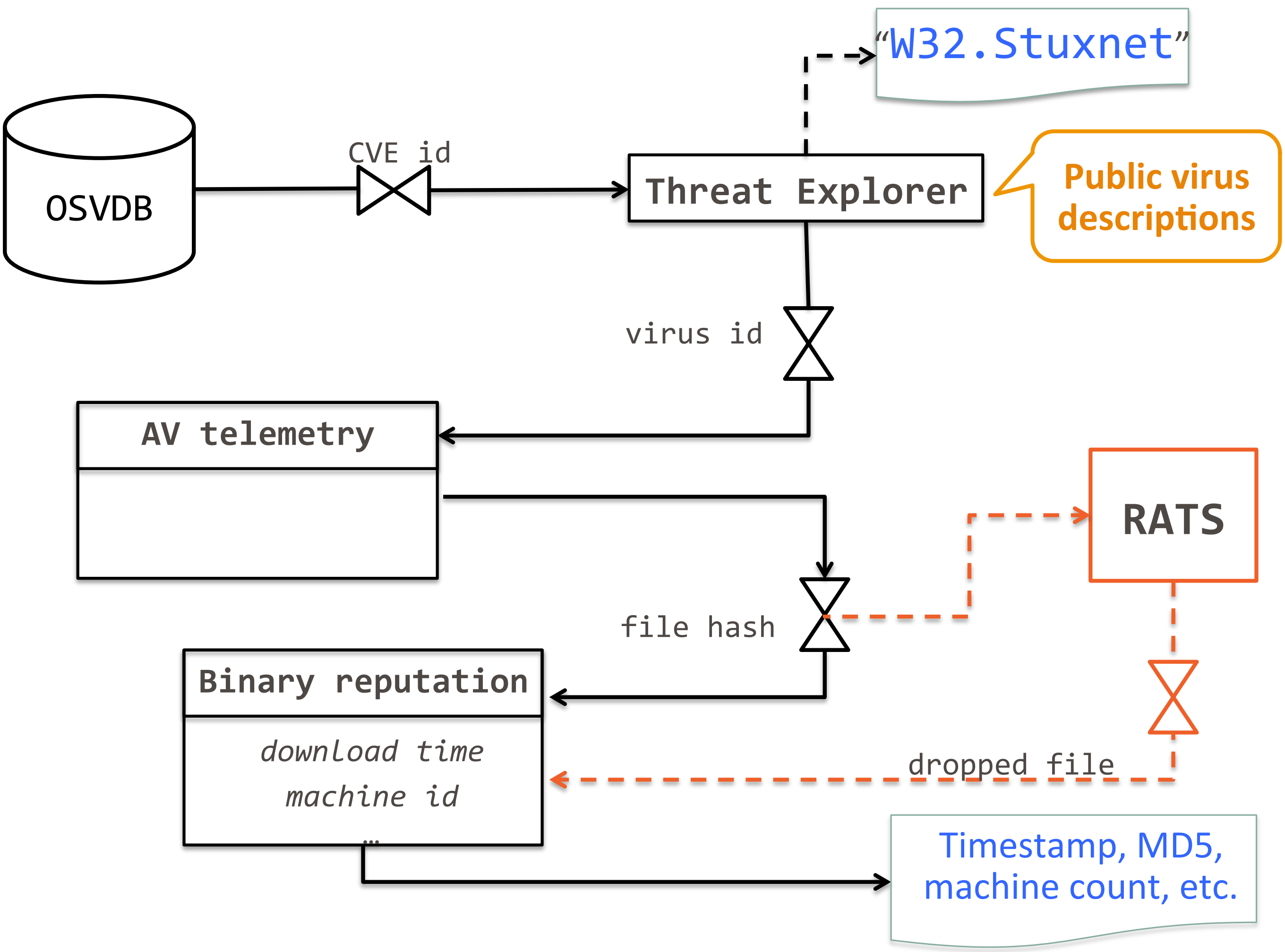
Spam: 2.5M decoys

WINE datasets for 0-day attack analysis

- Data collected since Dec 2009
- 225M detections
- 9M hosts

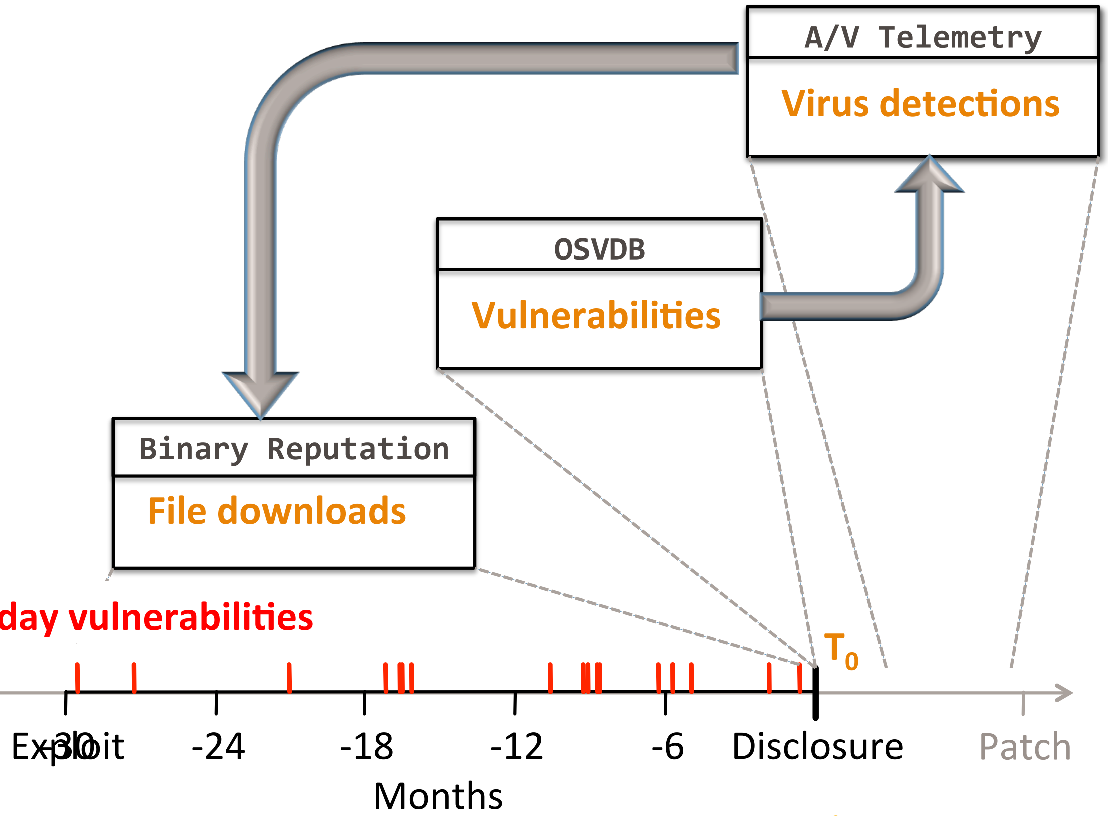
- Data collected since Feb 2008
- 32 Billion downloads
- 11M hosts
- 300M distinct files





Results

3 in 2008
7 in 2009
6 in 2010
2 in 2011

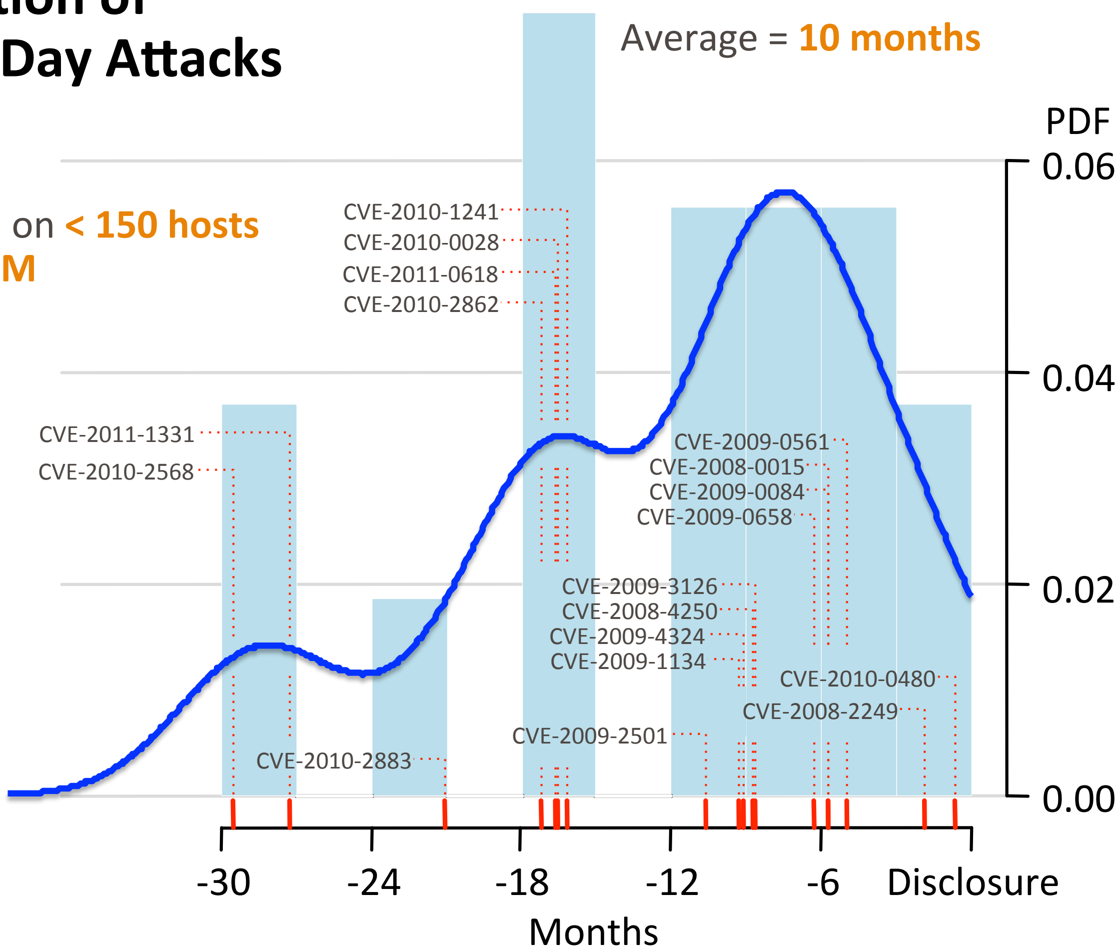


Found **18 zero-day vulnerabilities**
11 not known

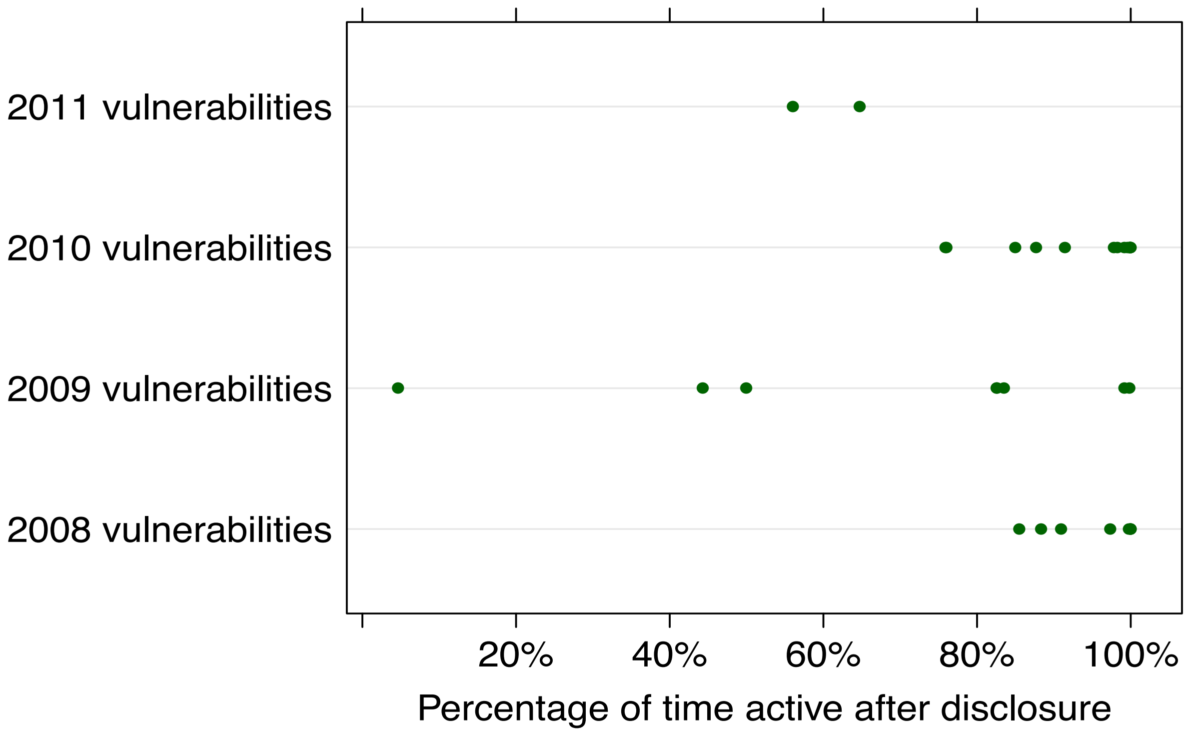
Duration of Zero-Day Attacks

Average = **10 months**

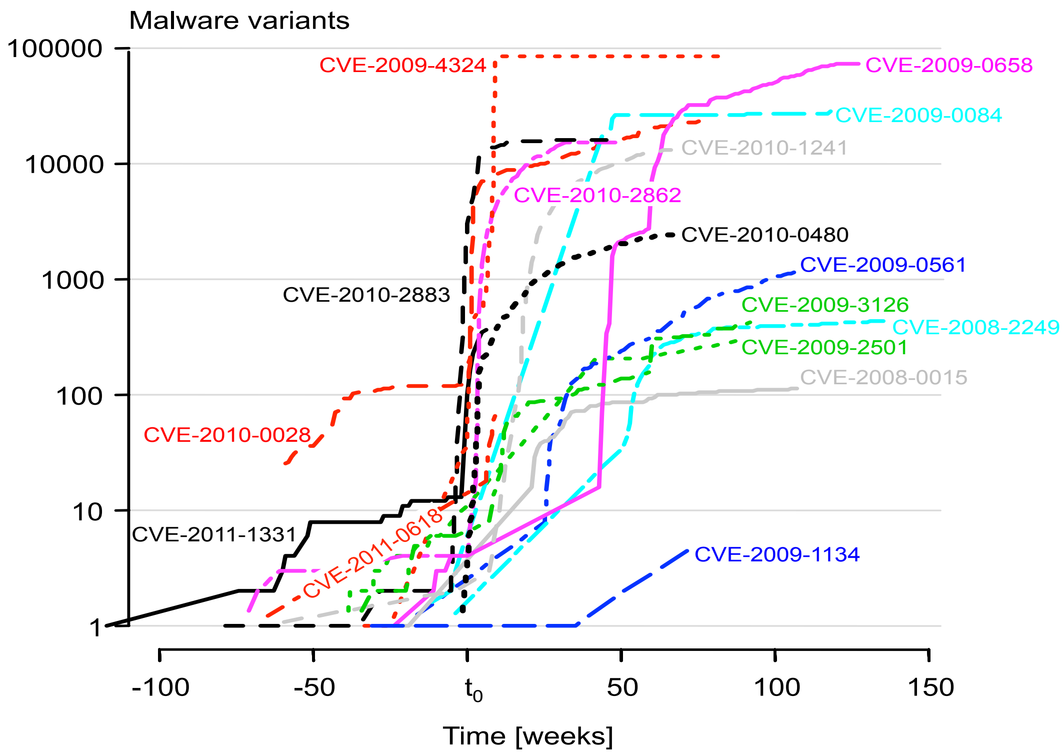
Detected on **< 150 hosts** out of **11M**



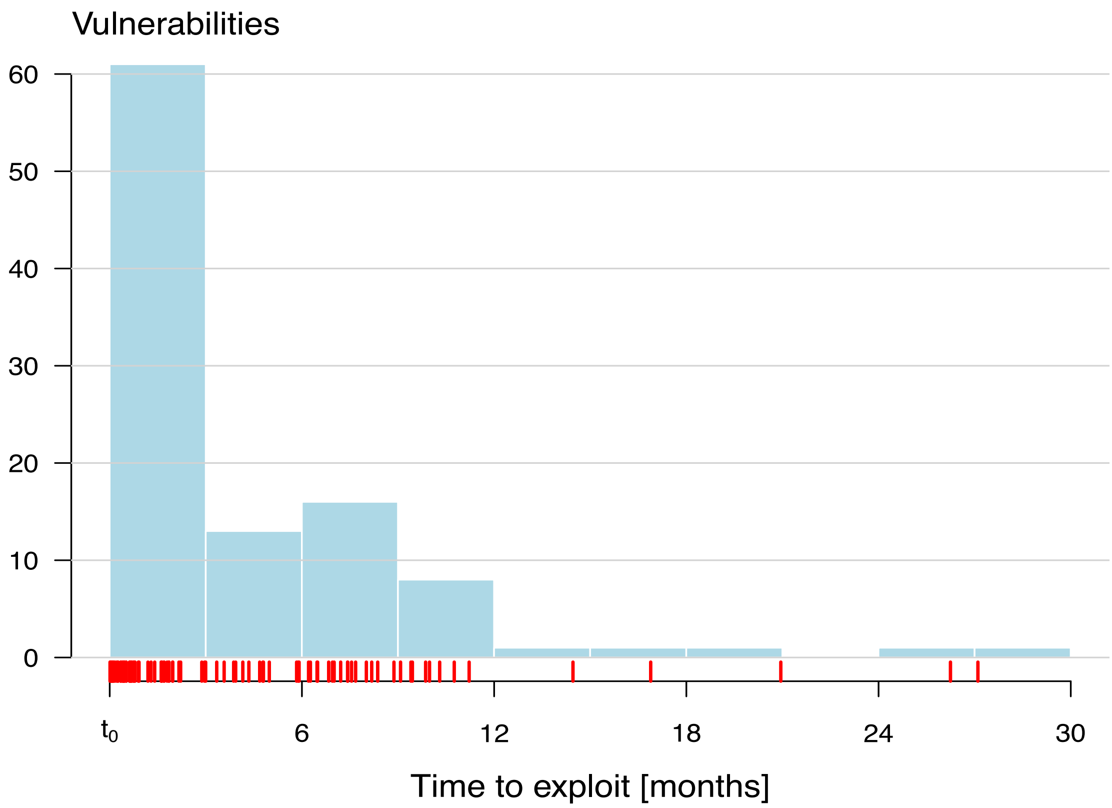
The usage of 0-day vulnerabilities after disclosure



Zero-day vulnerabilities after disclosure



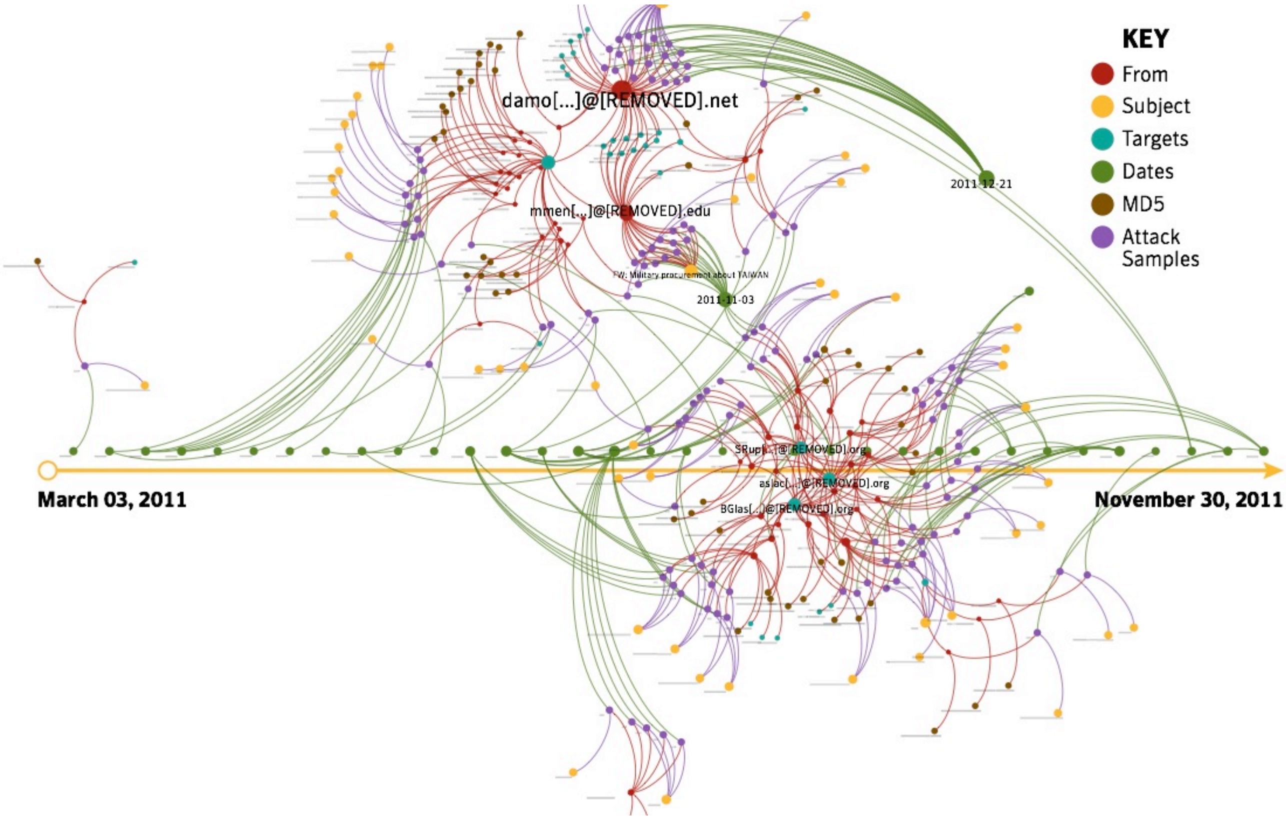
Reaction of the malware authors to the public disclosure



To disclose or not to disclose…

- Ongoing debate on the benefits of full disclosure policy
- Public disclosure provides an incentive for vendors to patch faster
- On the other hand, disclosing vulnerabilities causes an increase in the volume of attacks

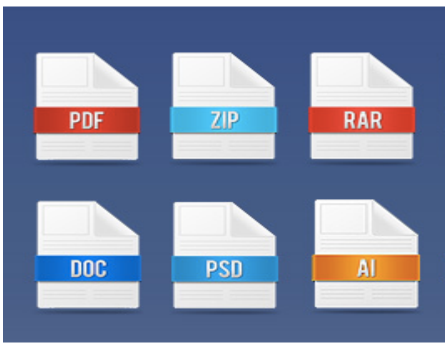
Taidoor Attacks - 2011



Limitations

Web attacks

Polymorphism



Exploits in non-executable files

Highly Targeted Attacks

Conclusion

- Using data collected from real users, we were able to find 18 zero-day vulnerabilities
- Zero-day attacks last between 19 days and 30 months, with a median of 8 months and an average of approximately 10 months
- The public disclosure of vulnerabilities is followed by an increase of up to five orders of magnitude in the volume of attacks
- To decrease the window of exposure, software vendors should be more careful to provide patches and make sure everyone applies them

Thank you!

Leylya_Yumer@symantec.com Tudor_Dumitras@symantec.com

http://www.symantec.com/WINE

2.1.3 Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting

Authors Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna.

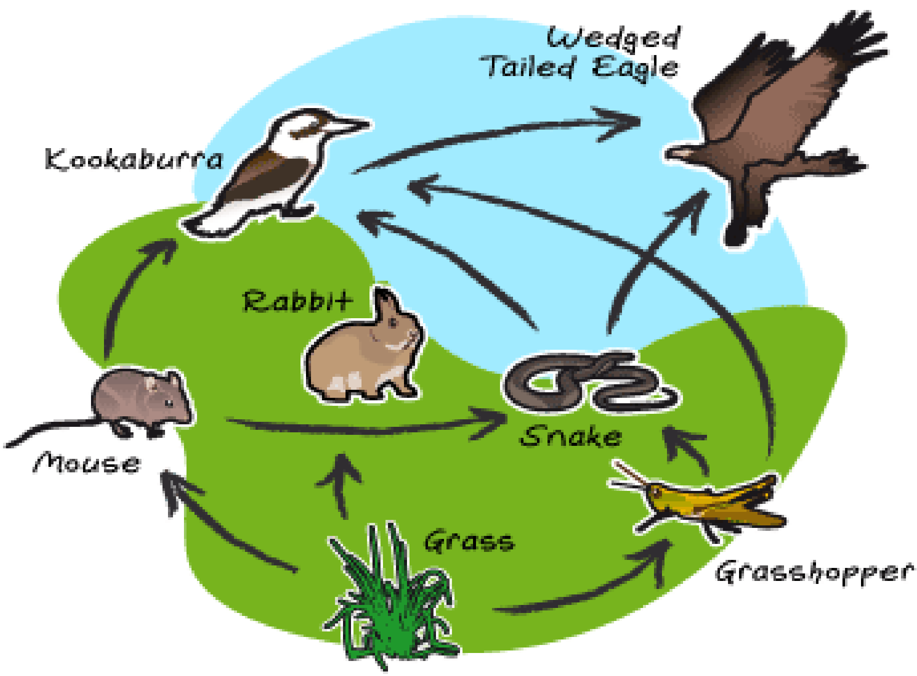
Speaker Nick Nikiforakis.

Paper Summary The web has become an essential part of our society and is currently the main medium of information delivery. Billions of users browse the web on a daily basis, and there are single websites that have reached over one billion user accounts. In this environment, the ability to track users and their online habits can be very lucrative for advertising companies, yet very intrusive for the privacy of users. In this paper, we examine how web-based device fingerprinting currently works on the Internet. By analyzing the code of three popular browser-fingerprinting code providers, we reveal the techniques that allow websites to track users without the need of client-side identifiers. Among these techniques, we show how current commercial fingerprinting approaches use questionable practices, such as the circumvention of HTTP proxies to discover a user's real IP address and the installation of intrusive browser plugins. At the same time, we show how fragile the browser ecosystem is against fingerprinting through the use of novel browser-identifying techniques. With so many different vendors involved in browser development, we demonstrate how one can use diversions in the browsers' implementation to distinguish successfully not only the browser-family, but also specific major and minor versions. Browser extensions that help users spoof the user-agent of their browsers are also evaluated. We show that current commercial approaches can bypass the extensions, and, in addition, take advantage of their shortcomings by using them as additional fingerprinting features.

Cookieless Monster

Exploring the Ecosystem of Web-based Device Fingerprinting

Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna



Motivation & Contributions

- Tracking involves more than just 3rd party cookies
- Fingerprinting: Telling users apart based on their browsing environments, without extra stateful identifiers
- Thorough study of current fingerprinting practices on the web
- Difficulty of hiding the true nature of a user's browsing environment

Users reacted…

- 1/3 of users delete first & third-party cookies within a month after they've been setup [8]
- Multiple extensions revealing hidden trackers
  - Ghostery
  - Collusion
- Private mode of browsers used to avoid traces of cookies from certain websites

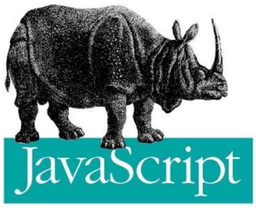
Advertisers reacted back…

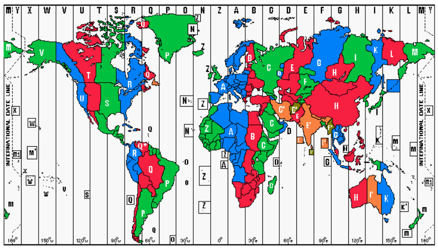
- What if you could track users without the need of cookies or any other stateful client-side identifier?
  - Hidden from users
  - Hard to avoid it / opt-out

Web-based device fingerprinting

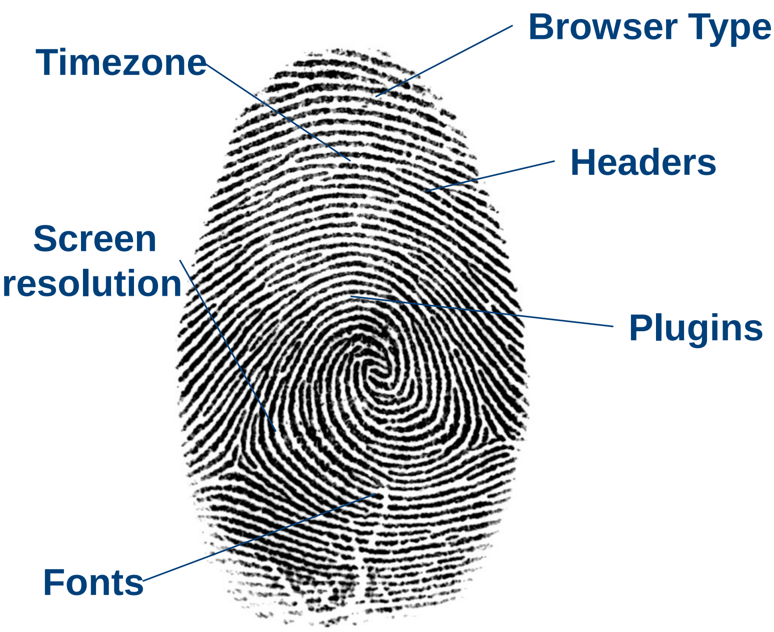
- Eckersley showed in 2010 that certain attributes of your browsing environment can be used to accurately track you
- These attributes, when combined, created a quite unique fingerprint of your system?
  - How?

Properties fingerprinted by Panopticlick





Resulting fingerprints



- 94.2% of the users with Flash/Java could be uniquely identified
- Simple heuristic algorithms could track updates of the same browser

Fast forward 2 years

- In mid 2012, all we knew is that fingerprinting is possible and that a small number of companies offer it as a service

- Questions that begged answering:
  - How are they doing it?
  - Could they do more?
  - Who is using them?
  - How are users trying to hide?
    - Is it working?

Manual analysis of 3 fingerprinting companies

bluecava where devices rule.

ThreatMetrix

iovation

1. Find the domains that they use to serve their fingerprinting scripts
2. Find some websites that use them and extract the code
3. De-obfuscate and analyze
4. Compare and classify

Step 3 took a while…

```
return;}var _i_b=_i_aa.getElementById(window.io_bbout_element_id);_i_b["value"]=_if_fa;}func
dow.io_bb_callback:__if_d;_i_c(_if_fa,_if_fb);}var _i_d={__if_p:function(_if_fc){return _if_
(_if_fc.getUTCDate(),2)+" "+this.__if_ad(_if_fc.getUTCHours(),2)+":"+this.__if_ad(_if_fc.get
 _i_e=_if_fd.toString(16);return(_i_m)?this.__if_ad(_i_e,_i_m):_i_e;},__if_u:function(_i_bz)
deAt(_i_g);if(_i_h>=56320&&_i_h<57344)continue;if(_i_h>=55296&&_i_h<56320){if(_i_g+1>=_i_bz.
nue;_i_h=((_i_h-55296)<<10)+(s-56320)+65536;}if(_i_h<128)_i_f+=String.fromCharCode(_i_h);els
f+=String.fromCharCode(224+(_i_h>>12),128+((_i_h>>6)&63),128+(_i_h&63));else _i_f+=String.fr
rn _i_f;},__if_y:function(_if_fe){if(typeof(encodeURIComponent)=="function")return encodeURI
length;_i_g++){var _i_k=_i_j.charAt(_i_g);var _i_l=new RegExp("[a-zA-Z0-9-_.!~*'()]");_i_f+=
nction(_i_bz,_if_ff){var _i_m="";var _i_n=_if_ff-_i_bz.length;while(_i_m.length<_i_n)_i_m+="
JKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",__if_aj:function(_i_bz){var _i_e="
i_bz.charCodeAt(_i_g+1);var _i_r=_i_bz.charCodeAt(_i_g+2);var _i_s=_i_p>>2;var _i_t=((_i_p&3
=64;}else if(isNaN(_i_r)){_i_v=64;}_i_e=_i_e+this._i_ej.charAt(_i_s)+this._i_ej.charAt(_i_t)
nction(_i_bz){var _i_w="";var _i_x,chr2,chr3="";var _i_s,_i_t,_i_u,_i_v="";var _i_g=0;var _i
indexOf(_i_bz.charAt(_i_g++));_i_t=this._i_ej.indexOf(_i_bz.charAt(_i_g++));_i_u=this._i_ej.
)|(_i_t>>4);chr2=((_i_t&15)<<4)|(_i_u>>2);chr3=((_i_u&3)<<6)|_i_v;_i_w=_i_w+String.fromCharC
ing.fromCharCode(chr3);_i_x=chr2=chr3="";_i_s=_i_t=_i_u=_i_v="";}while(_i_g<_i_bz.length);re
l:12,_i_em:false,_i_en:"",_i_eo:"",_i_ep:true};if(typeof(window.io_install_stm)!="boolean")w
.io_install_flash=_i_z._i_em;if(typeof(window.io_exclude_stm)!="number")window.io_exclude_st
b_url===undefined)window.io_stm_cab_url=_i_o.__if_aq("aHR0cHM6Ly9tcHNuYXJlLmllc25hcmUuY29t")
l_stm_error_handler===undefined)window.io_install_stm_error_handler=_i_z._i_en;if
needs_update_handler===undefined)window.io_flash_needs_update_handler=_i_z._i_eo;if(typeof(w
unction(_if_fg){if(_if_fg===undefined)return null;if(typeof(_if_fg)=="object"&&_if_fg.tagNam
tElementsByName(_if_fg);for(var _i_g=0;_i_g<_i_ab.length;_i_g++)if(_i_ab[_i_g]._i_dc&&_i_ab[
```


Results

- After extracting all features, we created a taxonomy of all fingerprinted features, and compared each company to Panopticlick
- Collectively, Panopticlick was fully covered

Panopticlick features	Additional features
Browser customizations	**ActiveX + CLSIDs**
Browser-level User Conf.	**DNT Choice**
Browser Family & Version	**Math constants**
OS & Applications	**Windows Registry**
Hardware & Network	**TCP/IP Parameters**

Non-trivial extras

- Non-plugin font detection
  - Comparison of text's width & height
- Native Fingerprinting plugins
  - Accessing highly-specific registry value
- Fingerprint delivery mechanisms
- Proxy detection

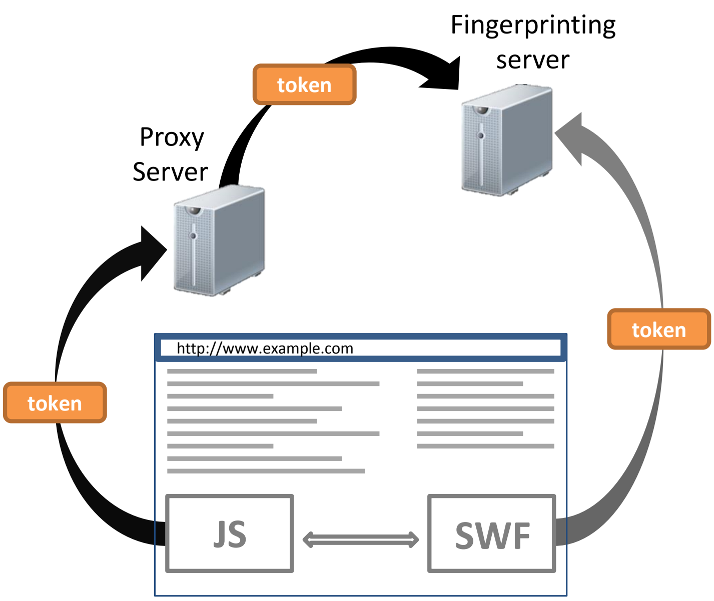
Font Detection through JavaScript

String	Dimensions
I_DO_NOT_NEED_FLASH	500 x 84
I_DO_NOT_NEED_FLASH	420 x 84
I_DO_NOT_NEED_FLASH	510 x 87
I_DO_NOT_NEED_FLASH	399 x 82

Non-trivial extras

- Non-plugin font detection
  - Comparison of text's width & height
- Native Fingerprinting plugins
  - Accessing highly-specific registry values
- Fingerprint delivery mechanisms
- Proxy detection

Proxy-detection



Adoption

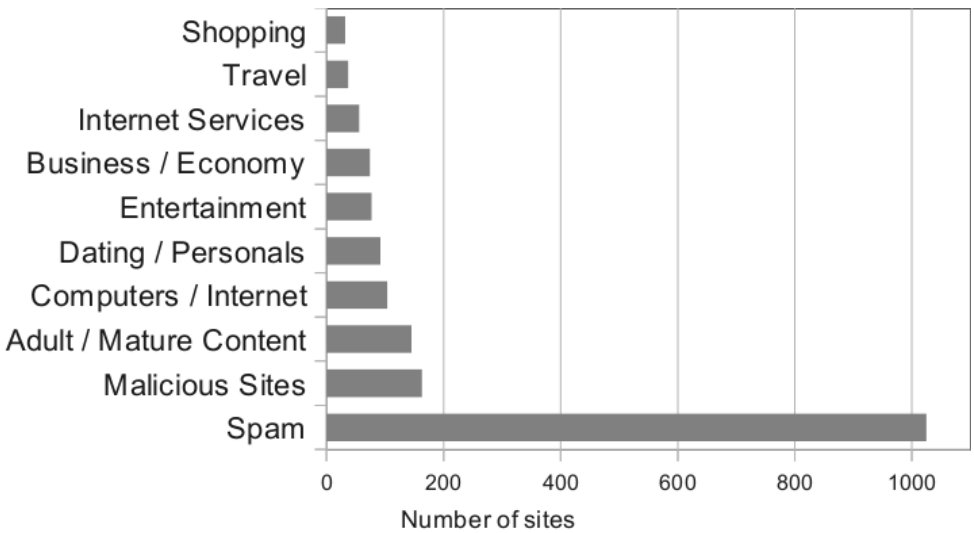
Dataset A

- Crawled top 10,000 sites, searching for inclusions from the 3 fingerprint providers
- 40 sites discovered
  - Porn & dating sites most prominent
    - Shared credentials & Sybil attacks
  - skype.com the highest ranking one

Adoption

Dataset B

- 3,804 domains from Wepawet



Status

- Fingerprinting is out there
  - Quite a number of new techniques over Panopticlick
- Large and popular sites are using them
- Could they be doing more?
  - How do the browser internals relate to a browser's identity?

UCSB

KU LEUVEN
DISTRINET RESEARCH GROUP

DIY Fingerprinting

UCSB

KU LEUVEN
DISTRINET RESEARCH GROUP

DIY Fingerprinting

- We decided to try some fingerprinting of our own
- Focus on ... erprinters
  probe the
  - naviga
  - screen
- Perform a ... search for
  difference
  - Add pr
  - Remov
  - Modify properties

Status

- Fingerprinting is out there
  - Quite a number of new techniques over Panopticlick
- Large and popular sites are using them
- There could be more fingerprinting done by the companies
- How could a user react?

Browser extensions

- Reviewed 11 different browser extensions that spoof a browser's user-agent
  - 8 Firefox + 3 Chrome
  - More than 800,000 users
- Advice to use such extensions:
  - Previous research in web tracking
  - Underground hacking guides
- How do they stand-up against fingerprinting?

Worse than nothing…

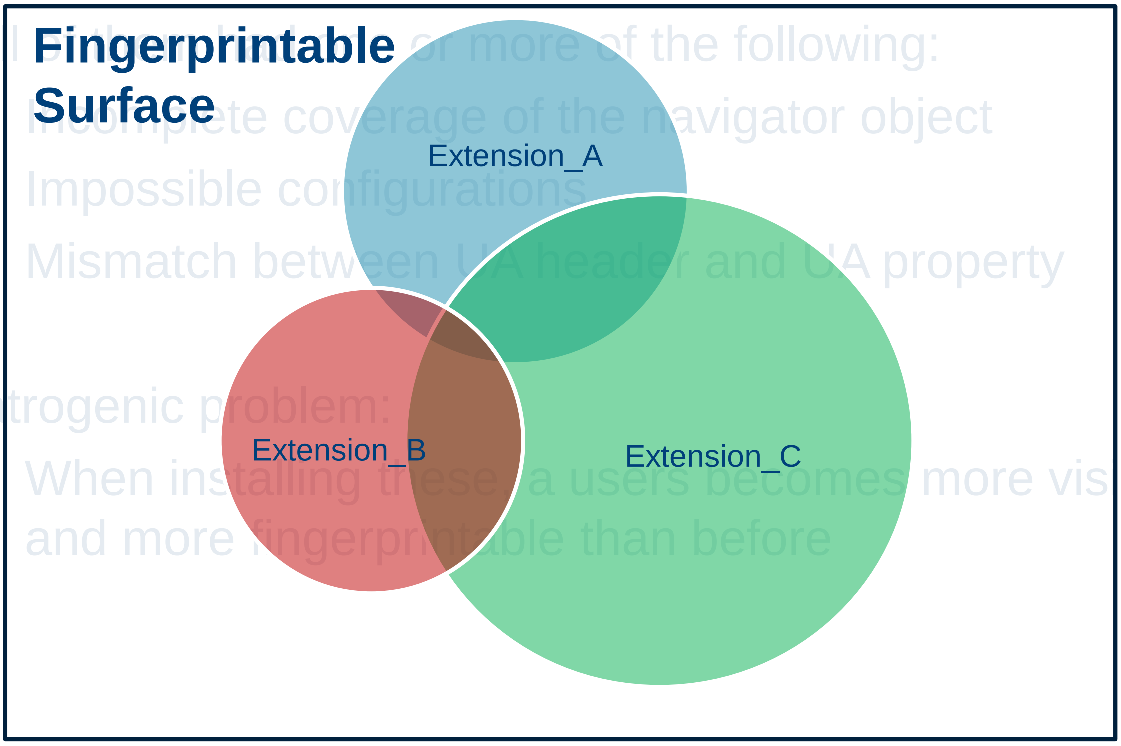
- All of them had one or more of the following:
  - Incomplete coverage of the navigator object
  - Impossible configurations
  - Mismatch between UA header and UA property
- Iatrogenic problem:
  - When installing these, a user becomes more visible and more fingerprintable than before

UCSB
KU LEUVEN
DISTRINET RESEARCH GROUP

Worse than nothing…

- A[illegible] or more of the following:
  - Incomplete coverage of the navigator object
  - Impossible configurations
  - Mismatch between UA header and UA property

- Iatrogenic problem:
  - When installing these, a users becomes more visible and more fingerprintable than before



History and tips

- Paper was accepted on the 1st try
  - So, not so many lessons learnt

- General guidelines
  - Topic is really important
  - Try to look at your problem as part of a greater whole, i.e. expand horizontally
  - Polish, polish, polish
  - Do good work ☺

UCSB
KU LEUVEN
DISTRINET RESEARCH GROUP

Conclusion

- Fingerprinting is a real problem
- Browsers are so complex that it is really hard to make them seem identical
- Current browser extensions should not be used for privacy reasons
- Long term solutions will most-likely not be pure technical ones
  - Legislation required, like in stateful tracking

Thank you

nick.nikiforakis@cs.kuleuven.be
http://www.securitee.org

2.1.4 Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security

Authors Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, Lars Baumgartner, Bernd Freisleben

Speaker Sascha Fahl.

Paper Summary Many Android apps have a legitimate need to communicate over the Internet and are then responsible for protecting potentially sensitive data during transit. This paper seeks to better understand the potential security threats posed by benign Android apps that use the SSL/TLS protocols to protect data they transmit. Since the lack of visual security indicators for SSL/TLS usage and the inadequate use of SSL/TLS can be exploited to launch Man-in-the-Middle (MITM) attacks, an analysis of 13,500 popular free apps downloaded from Google's Play Market is presented. We introduce MalloDroid, a tool to detect potential vulnerability against MITM attacks. Our analysis revealed that 1,074 (8.0%) of the apps examined contain SSL/TLS code that is potentially vulnerable to MITM attacks. Various forms of SSL/TLS misuse were discovered during a further manual audit of 100 selected apps that allowed us to successfully launch MITM attacks against 41 apps and gather a large variety of sensitive data. Furthermore, an online survey was conducted to evaluate users' perceptions of certificate warnings and HTTPS visual security indicators in Android's browser, showing that half of the 754 participating users were not able to correctly judge whether their browser session was protected by SSL/TLS or not. We conclude by considering the implications of these findings and discuss several countermeasures with which these problems could be alleviated.

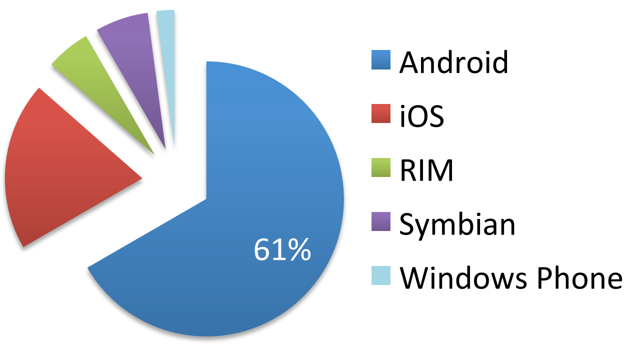
Why Eve and Mallory Love Android
An Analysis of Android SSL (In)Security

Sascha Fahl
Marian Harbach
Thomas Muders
Lars Baumgärtner
Bernd Freisleben
Matthew Smith

Some Android Facts

- 750 million devices (as of Q1 2013)
- > 1 million activations per day (as of Q2 2013)
- 750,000 apps (as of Q2 2013)

Market Share (Q1 2013)



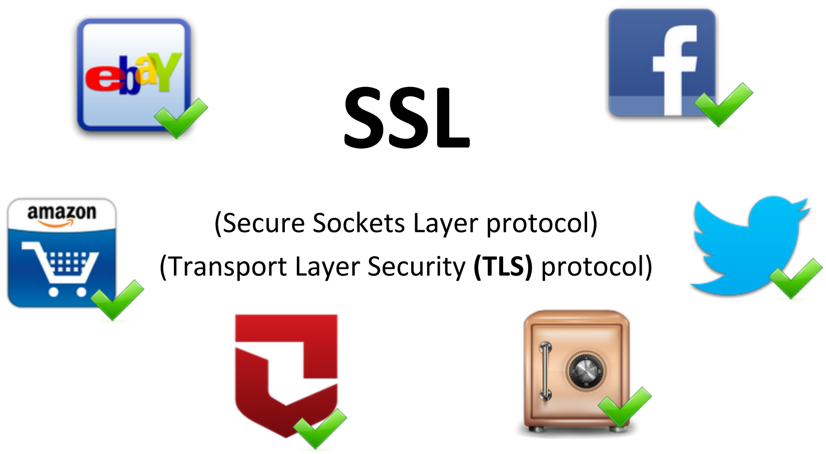
Appification

- There's an App for Everything

What do Most Apps Have in Common?

They share data over the Internet

Some of them secure transfer using:



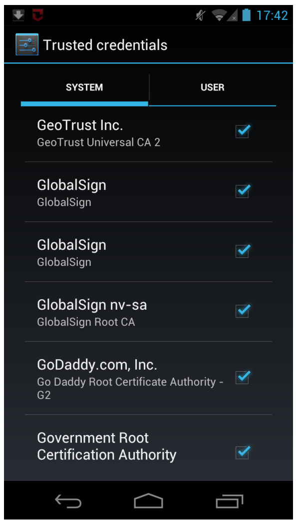
SSL Usage on Android

The default Android API implements correct certificate validation.

What could possibly go wrong?

SSL Usage on Android

- A server needs a certificate that was signed by a trusted Certificate Authority (~130 pre-installed CAs)
- For non-trusted certificates a custom workaround is needed



What about using a non-trusted certificate?

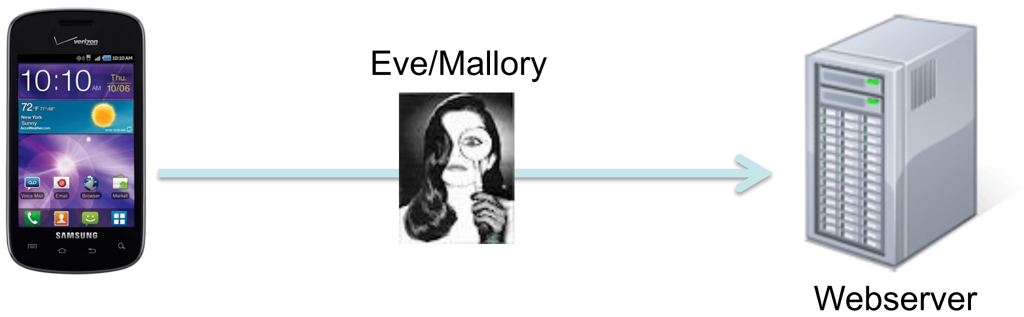
Q: Does anyone know how to accept a self signed cert in Java on the Android? A code sample would be perfect.
A: Use the EasyX509TrustManager library hosted on code.google.com.

Q: I am getting an error of „javax.net.ssl.SSLException: Not trusted server certificate“. I want to simply allow any certificate to work, regardless whether it is or is not in the Android key chain. I have spent 40 hours researching and trying to figure out a workaround for this issue.
A: Look at this tutorial http://blog.antoine.li/index.php/2010/10/android-trusting-ssl-certificates

stackoverflow.com

Our Analysis

- downloaded 13,500 popular and free Apps from Google’s Play Market
- built MalloDroid which is an androguard extension to analyze possible SSL problems in Android Apps
  - broken TrustManager implementations
  - accept all Hostnames

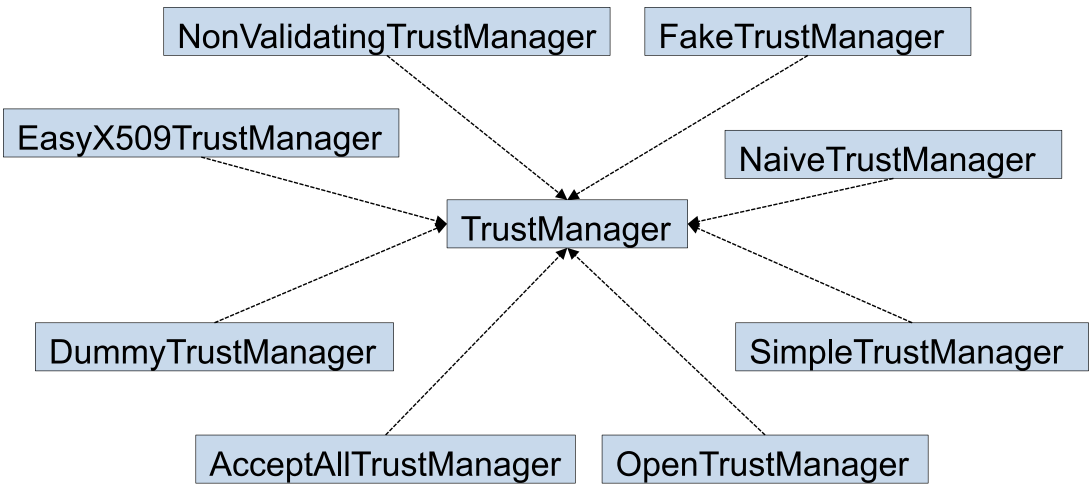


Static Code Analysis Results

- 92,8 % Apps use INTERNET permission
- 91,7 % of networking API calls HTTP(S) related
- 0,8 % exclusively HTTPS URLs
- 46,2 % mix HTTP and HTTPS
- 17,28 % of all Apps that use HTTPS include code that fails in SSL certificate validation
  - 1070 include critical code
  - 790 accept all certificates
  - 284 accept all hostnames

TrustManager Implementations

- 22 different TrustManager implementations



- and all turn effective certificate validation off

Manual App Testing Results

- cherry-picked 100 Apps
- 21 Apps trust all certificates
- 20 Apps accept all hostnames

What we found:

Manual App Testing Results

39 – 185 million affected installs!

What we found:

One Example

Zoner AV

AV-TEST
The Independent IT-Security Institute
Magdeburg Germany

- Anti-Virus App for Android
- Awarded best free Anti-Virus App for Android by av-test.org

Zoner AV

- Virus signature updates via HTTPS GET
- No check for the update's authenticity!
- The good thing: It uses SSL
  - Unfortunately: The wrong way

```
static final HostnameVerifier DO_NOT_VERIFY = new HostnameVerifier()
{
        public boolean verify(String paramString, SSLSession paramSSLSession)
        {
          return true;
        }
};
```

Zoner AV

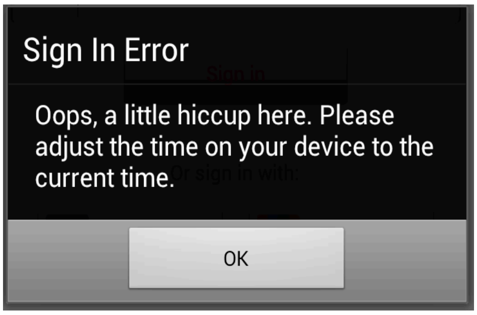
- We did the following

More Examples

- Remote Control App
- Remote Code Injection
- Unlocking Rental Cars

How Do (Good) Apps React to MITMAs?

- Technically
- Usability



Flickr

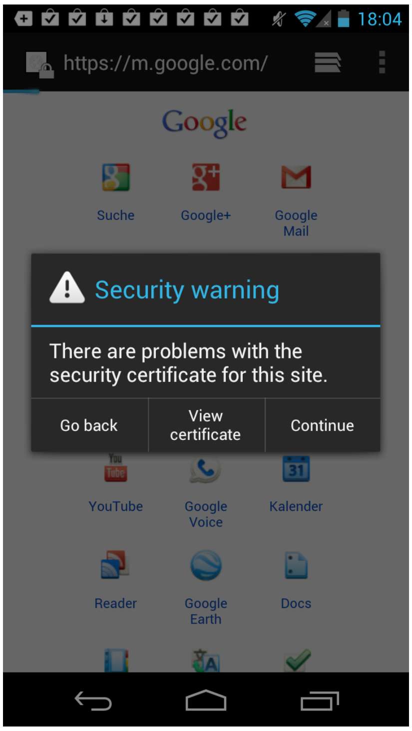
Facebook

Browser Warning Messages

All do SSL certificate validation correctly…

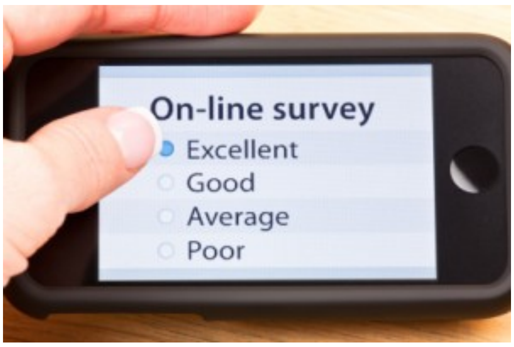
… and warn the user if something goes wrong….

SSL Warning Messages – Android Stock Browser



Online Survey

- To find out if the Browser's warning messages help the users
  - presented an SSL warning message
- To see if users know when they are surfing on an SSL protected website
  - half of the participants HTTP
  - half of the participants HTTPS



Online Survey - Results

- 745 participants
- 47.5% of non-IT experts believed they were using a secure Internet connection...although it was plain HTTP.
- ~50% had not seen an SSL warning message on their phone before.
- The risk users were warned against was rated with 2.86 (sd=.94) on a scale between 1 and 5
- Many participants stated they did not care about warning messages at all.

How can we protect the user?

Rethinking SSL Development in an Appified World, CCS'13

2.1.5 Don't trust satellite phones: a security analysis of two satphone standards

Authors Benedikt Driessen, Ralf Hund, Carsten Willems, Christof Paar, Thorsten Holz.

Speaker Benedikt Driessen.

Paper Summary There is a rich body of work related to the security aspects of cellular mobile phones, in particular with respect to the GSM and UMTS systems. To the best of our knowledge, however, there has been no investigation of the security of satellite phones (abbr. sat phones). Even though a niche market compared to the G2 and G3 mobile systems, there are several 100,000 sat phone subscribers worldwide. Given the sensitive nature of some of their application domains (e.g., natural disaster areas or military campaigns), security plays a particularly important role for sat phones. In this paper, we analyze the encryption systems used in the two existing (and competing) sat phone standards, GMR-1 and GMR-2. The first main contribution is that we were able to completely reverse engineer the encryption algorithms employed. Both ciphers had not been publicly known previously. We describe the details of the recovery of the two algorithms from freely available DSP-firmware updates for sat phones, which included the development of a custom disassembler and tools to analyze the code, and extending prior work on binary analysis to efficiently identify cryptographic code. We note that these steps had to be repeated for both systems, because the available binaries were from two entirely different DSP processors. Perhaps somewhat surprisingly, we found that the GMR-1 cipher can be considered a proprietary variant of the GSM A5/2 algorithm, whereas the GMR-2 cipher is an entirely new design. The second main contribution lies in the cryptanalysis of the two proprietary stream ciphers. We were able to adopt known A5/2 cipher text-only attacks to the GMR-1 algorithm with an average case complexity of 2^{32} steps. With respect to the GMR-2 cipher, we developed a new attack which is powerful in a known-plaintext setting. In this situation, the encryption key for one session, i.e., one phone call, can be recovered with approximately 50-65 bytes of key stream and a moderate computational complexity. A major finding of our work is that the stream ciphers of the two existing satellite phone systems are considerably weaker than what is state-of-the-art in symmetric cryptography.

An Experimental Security Analysis of Two Satphone Standards

Benedikt Driessen

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany

Summer School on RE, Bochum, Germany
24.07.2013

Acknowledgment

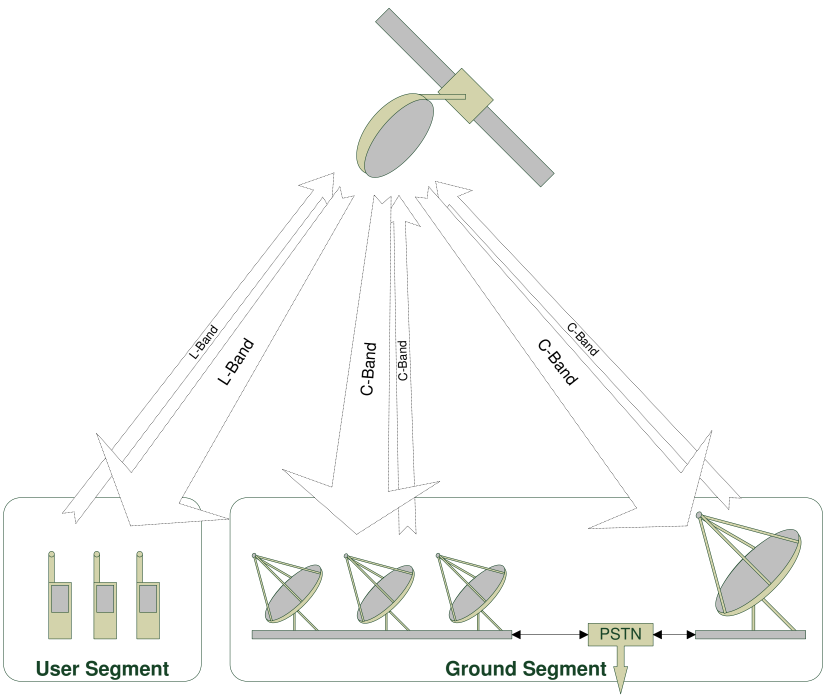
Joint work with several people, all from HGI:

- Ralf Hund
- Carsten Willems
- Christof Paar
- Thorsten Holz

Why analyze GMR-1 and GMR-2?

- Reasons for using satphones instead of cellphones
  - Cellphone infrastructure not always *available*
    - Oil rigs, ships, airplanes, deserts, poles
  - Cellphones not always *desirable*, e.g. in "rouge states"
    - Attacks public for more than 10 years
    - Locating handsets is easy
    - GSM infrastructure often accessible by local government
- GMR-1 and GMR-2 are major standards
  - Estimated user base: 350k – 500k *active* users
  - TerreStar and SkyTerra currently implement GMR-1
  - Specifications public, ciphers treated as black boxes
- **What is the security level provided by GMR-based systems?**

Network architecture



What we knew (and conjectured)

- GMR-1 and GMR-2 are derived from GSM
  - Ciphers are named A5-GMR-1 and A5-GMR-2 (GSM: A5/x)
  - Session based encryption (e.g. one key per call)
    - Challenge-and-response protocol involving secret on SIM card
- Typical satphone is made up of two processors
  - General purpose CPU (e.g. ARM) running some embedded OS
  - Specialized DSP for encoding, modulation, signal processing
  - ARM responsible for extracting and initializing DSP firmware
  - Encryption part of encoding process and *probably* done on DSP

Our approach

- Unknown ciphers are responsible for security of GMR
  - Satphones need to implement and execute ciphers
  - Ciphers can be obtained from satphone software
- **Perform cryptanalysis to assess security level**
- Procedure to find ciphers in software
  1. Choose appropriate satphone and obtain firmware
  2. Dissect firmware, locate DSP initialization in ARM code
  3. Reconstruct and dump DSP code
  4. Disassemble DSP code
  5. Find encryption algorithm
  6. Translate algorithm to C code and diagrams

GMR-1

Analyzing Thuraya's firmware

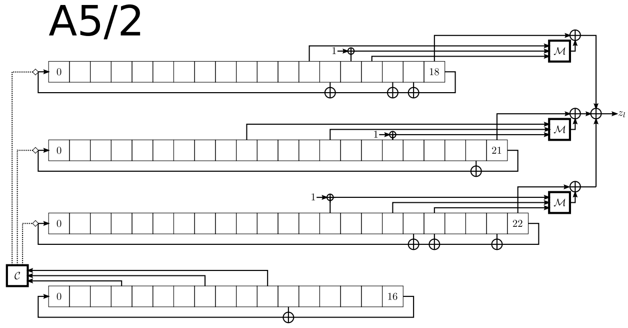
- Thuraya SO-2510 (ARM + TI C55x DSP)
  - Downloaded firmware update from Thuraya's website
  - IDA to find DSP initialization
  - QEMU to execute initialization routine
  - IDA to analyze reconstructed DSP firmware
    - Static analysis of 240kB of DSP code
    - No symbols, strings or other clues

Finding A5-GMR-1

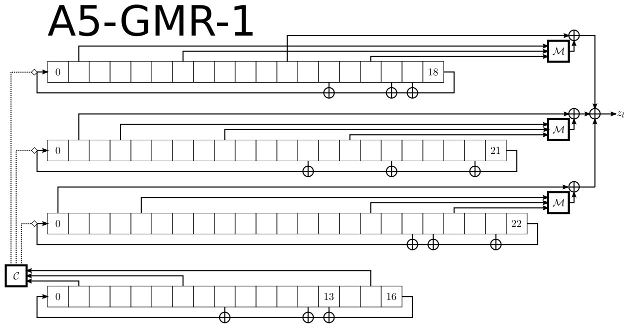
- Assumption: A5-GMR-1 might bear some resemblance to A5/1 or A5/2
  - GMR standards are derived from GSM
  - A5/x based on Linear Feedback Shift Registers (LFSRs)
  - LFSRs require a lot of XORing and SHIFTing
- **Idea**: Apply heuristics to find cipher (Caballero'09)
  - Rank functions by percentage of XOR/SHIFT operations
  - Four top ranked functions (35%–40% of XOR/SHIFT) adjacent in memory
  - Each function implements one LFSR of A5-GMR-1

A5-GMR-1 is a variant of A5/2

A5/2



A5-GMR-1

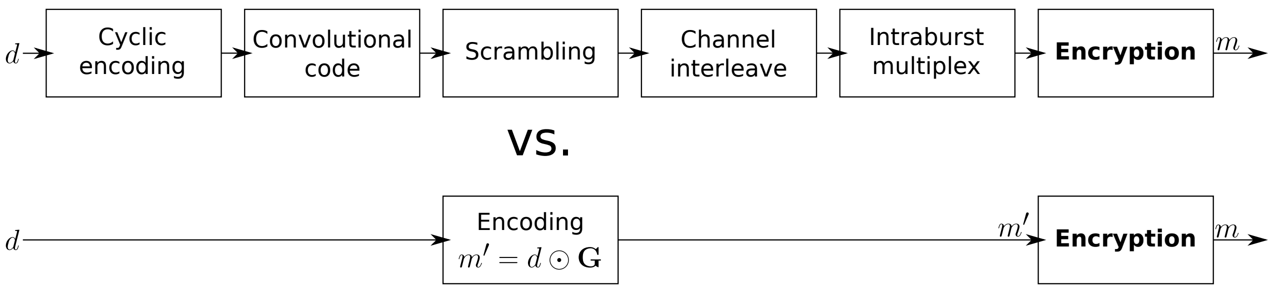


- A5-GMR-1 is based on A5/2
  - Feedback (and output taps) polynomials were changed
  - Initialization process slightly changed
- GSM attacks can be adapted
  - Known-plaintext attack (Petrovic'00)
  - Ciphertext-only attack (Barkan'03)

From a known keystream attack ..

- The clocking of the registers $R1 - R3$ is determined by $R4$
- Classical guess-and-determine attack
  - Guess $R4$ and clock cipher to obtain quadratic equations
  - Linearize equations to obtain $\mathbf{A} \odot x = z$
  - Solve equation system and test state candidate x
  - Obtain potential key from x and test it
- Known keystream (or plaintext) is limited in GMR

.. to a ciphertext-only ..

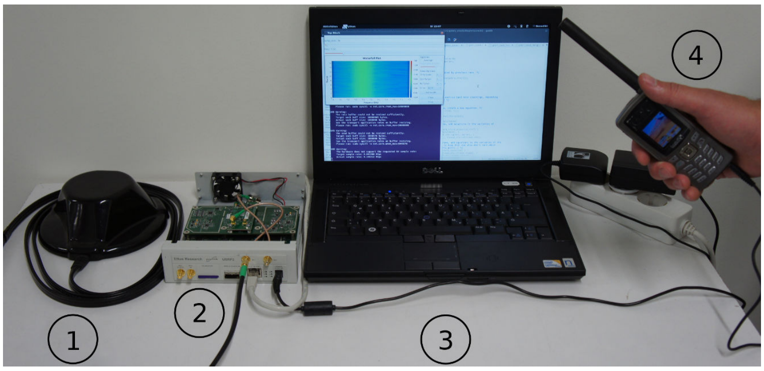


- Encoding is done prior to encryption
  - If we don't know d, we still know something about the structure of m'
- Encoding is linear
  - Encoding d into m' is a linear operation, i.e., $m' = d \odot \mathbf{G}$
  - Encrypting m' into m is also linear, $m = m' \oplus z$

.. attack on A5-GMR-1

- In a ciphertext-only attack scenario we have $m = \overbrace{(d \odot \mathbf{G})}^{m'} \oplus z$
  - $\mathbf{G}$ can be computed from the GMR specifications
  - d and z are unknown
- Exploit encoding to enable an efficient ciphertext-only attack
  - Construct parity check matrix $\mathbf{H}$ with $\mathbf{H} \odot m' = 0$
  - Use $\mathbf{H}$ to "cancel out" plaintext from ciphertext bits
- Attack similar to known-plaintext attack, but now we generate and solve $(\mathbf{H} \odot \mathbf{A}) \odot x = \mathbf{H} \odot m$

Results of attacking the Thuraya network



- Real-world attack reveals session key in a few minutes
  - Equipment for \$5,000 (Thuraya SO-2510, USRP-2, antenna, laptop) to capture downlink data
  - GNURadio, OsmocomGMR and some custom code to demodulate, decode and cryptanalyze captured data
  - 2^{21} guesses and 16 frames of TCH3 speech data required

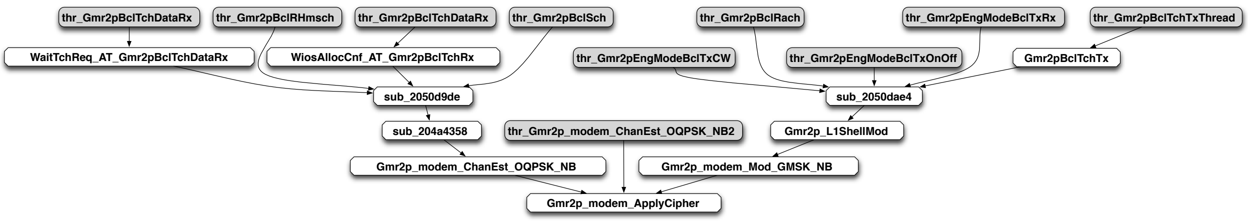
GMR-2

Analyzing Inmarsat's firmware

- IsatPhone Pro (ARM + AD Blackfin DSP)
  - Downloaded firmware from Inmarsat's website
  - IDA to analyze firmware updater
  - IDA script to reconstruct DSP image
  - Custom disassembler to disassemble Blackfin code
    - Static analysis of 300k lines of DSP code
    - Custom code for generation of callgraphs
    - Manual identification of arithmetic functions (`div32`/`rem32`/etc.)

`ApplyCipher` as start of our Odyssey

- Ranking approach did not work
- Inmarsat left names of source files in binary
  - Identify functions by source file names
  - `../modem/internal/Gmr2p_modem_ApplyCipher.c`
- `ApplyCipher` XORs two buffers
  - Backtracking input params too complex
- Reverse callgraph reveals ten thread functions



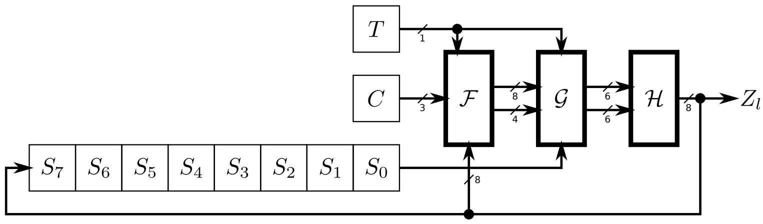
Motivation & Background
Motivation & Background
Analysis
Conclusions
GMR-1
GMR-2

Finding A5-GMR-2

- Thread functions implement state machines
  - Allocation of zero'ed keystream buffer in initial state
  - Call to `ApplyCipher` in later state
  - Call to cipher must happen in between
- **Idea**: Intersect set of all functions called by these threads
  - Found 13 shared sub-callgraphs
  - Cipher was then found manually

Benedikt Driessen
An Analysis of GMR-1 and GMR-2

A5-GMR-2 is ... different



- A5-GMR-2 is a byte oriented stream cipher with memory
  - 3-bit counter C, 1-bit counter T
  - $\mathcal{F}$ combines two bytes of session key with previous output
  - $\mathcal{G}$ is used for mixing purposes
  - $\mathcal{H}$ consists of two DES Sboxes as nonlinear output filter

A known-plaintext attack

- Exploit property of “keyschedule” in A5-GMR-2 to obtain an efficient known-plaintext attack
  - Given one of the two selected keybytes, the second can be determined from keystream
- **Result**: Efficient attack with keystream/time trade-off
  - Given 50–65 bytes of keystream, session key found after 2^{18} operations
  - Given 200 bytes of keystream, 2^{10} operations

Summary

- A5-GMR-1 and A5-GMR-2 reverse engineered from firmware updates
  - Ciphers were independently verified

Motivation & Background
Motivation & Background
Analysis
Conclusions

Summary

- A5-GMR-1 and A5-GMR-2 reverse engineered from firmware updates
  - Ciphers were independently verified
- Both ciphers were completely broken
  - Efficient ciphertext-only attack on GMR-1
  - Efficient known-plaintext attack on GMR-2

Summary

- A5-GMR-1 and A5-GMR-2 reverse engineered from firmware updates
  - Ciphers were independently verified
- Both ciphers were completely broken
  - Efficient ciphertext-only attack on GMR-1
  - Efficient known-plaintext attack on GMR-2
- **ETSI satellite communication standards offer no real privacy**

Lessons learned

- Although satellite communication is considered a niche market, some use cases are highly critical
  - Don't trust satellite phones in critical use cases!
  - Use additional layers of encryption
- Our effort was significant, but it could have been a lot harder
  - Don't make your *complete firmware* available for download
  - Strip useless strings from binaries
  - Apply some basic obfuscation techniques (packers, string obfuscation)
- Security through obscurity is still no good

Thanks

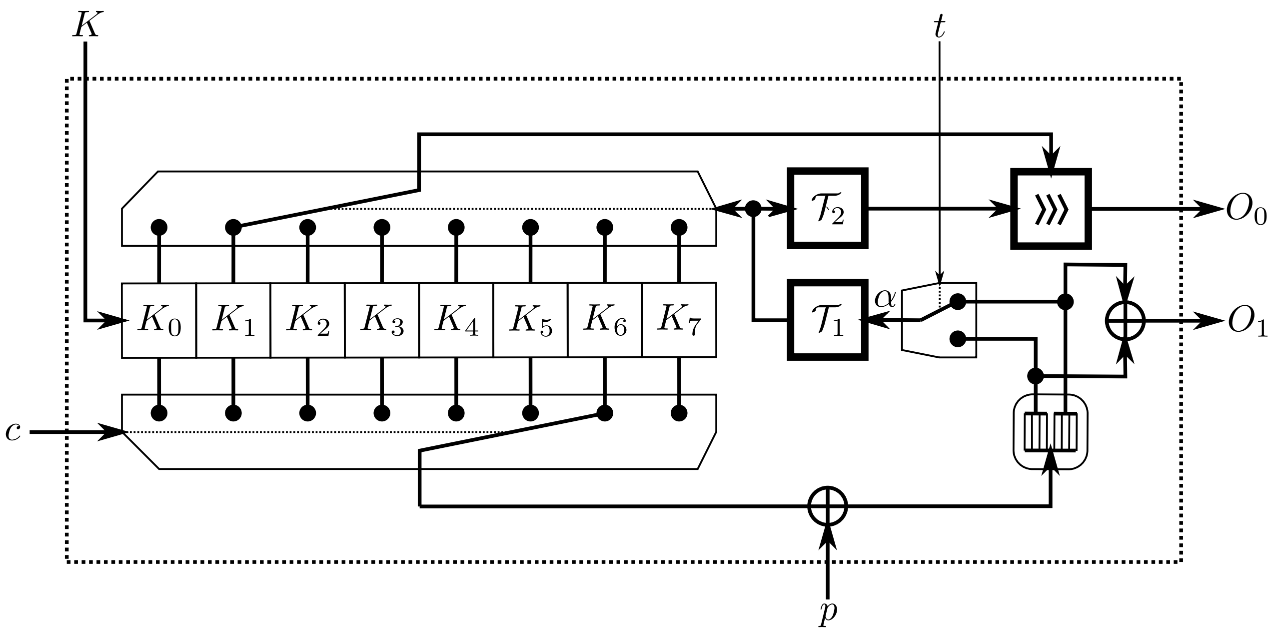
Thank you for your attention!

Any questions?

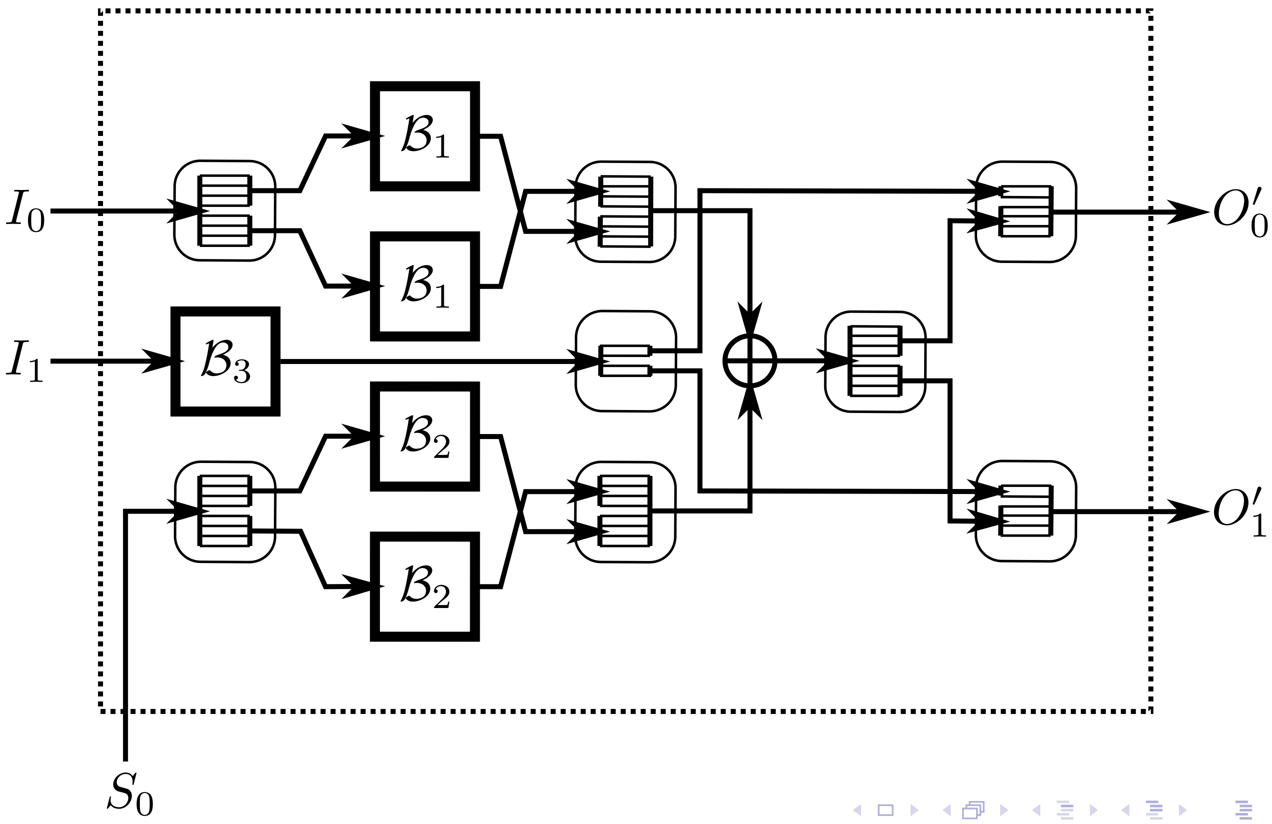
Motivation & Background
Motivation & Background
Analysis
Conclusions

Benedikt Driessen An Analysis of GMR-1 and GMR-2

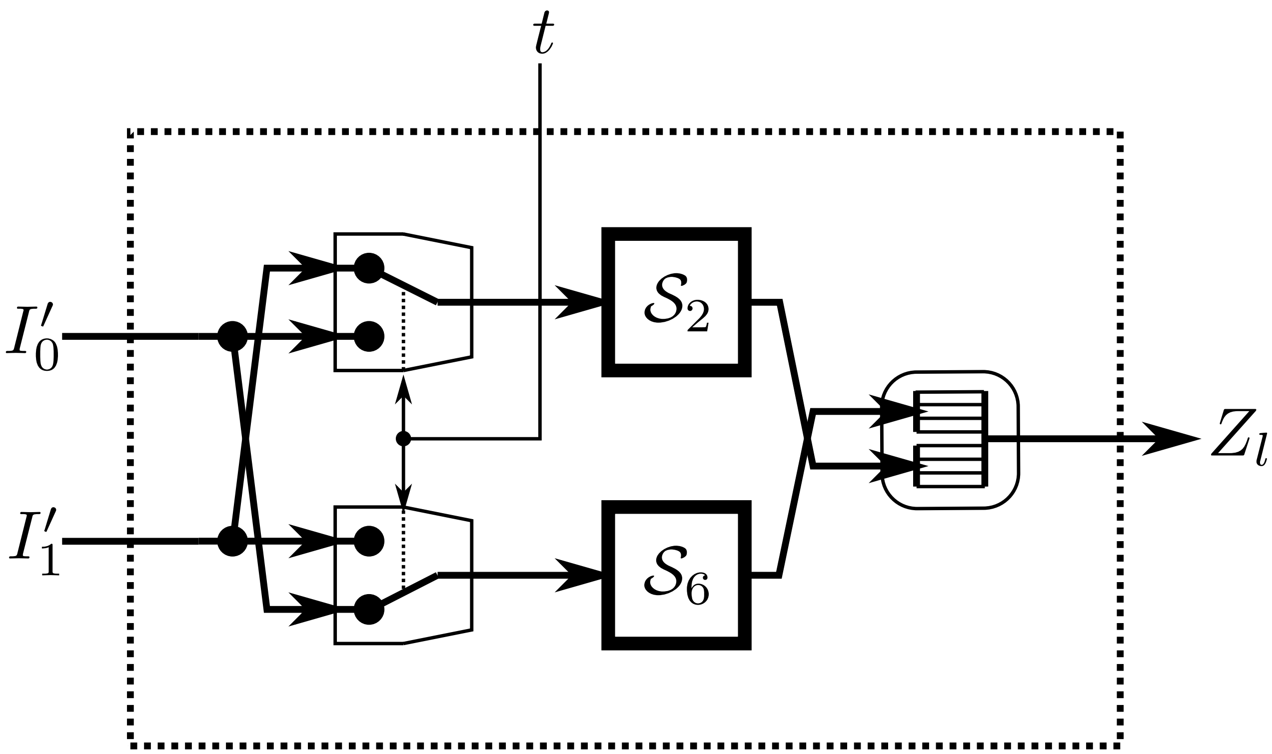
A5-GMR-2: The $\mathcal{F}$ function



A5-GMR-2: The $\mathcal{G}$ function



A5-GMR-2: The $\mathcal{H}$ function



A ciphertext-only attack on A5-GMR-1

- From a known-plaintext attack...
  - Guess $R4$ and clock cipher to obtain quadratic equations
  - Linearize equations to obtain $\mathbf{A} \odot x = z$
  - Solve equation system and test state candidate x
- ..to a ciphertext-only attack
  - Encoding d into m' is a linear operation, i.e., $m' = d \odot \mathbf{G}$
  - Encrypting m' into m is also linear, $m = m' \oplus k$
  - Construct parity check matrix $\mathbf{H}$ with $\mathbf{H} \odot m' = 0$
  - Use $\mathbf{H}$ to "cancel out" plaintext from ciphertext bits

$$\begin{aligned}\mathbf{H} \odot m &= \mathbf{H} \odot (m' \oplus z)\\ &= \underbrace{\mathbf{H} \odot m'}_{=0} \oplus \mathbf{H} \odot z\\ &= \underbrace{\mathbf{H} \odot \mathbf{A}}_{\mathbf{S}} \odot x = \mathbf{S} \odot x\end{aligned}$$

A known-plaintext attack on A5-GMR-2

- Too involved, please read paper.

2.1.6 Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization

Authors Alex Biryukov, Ivan Pustogarov, Ralf-Philipp Weinmann.

Speaker Alex Biryukov.

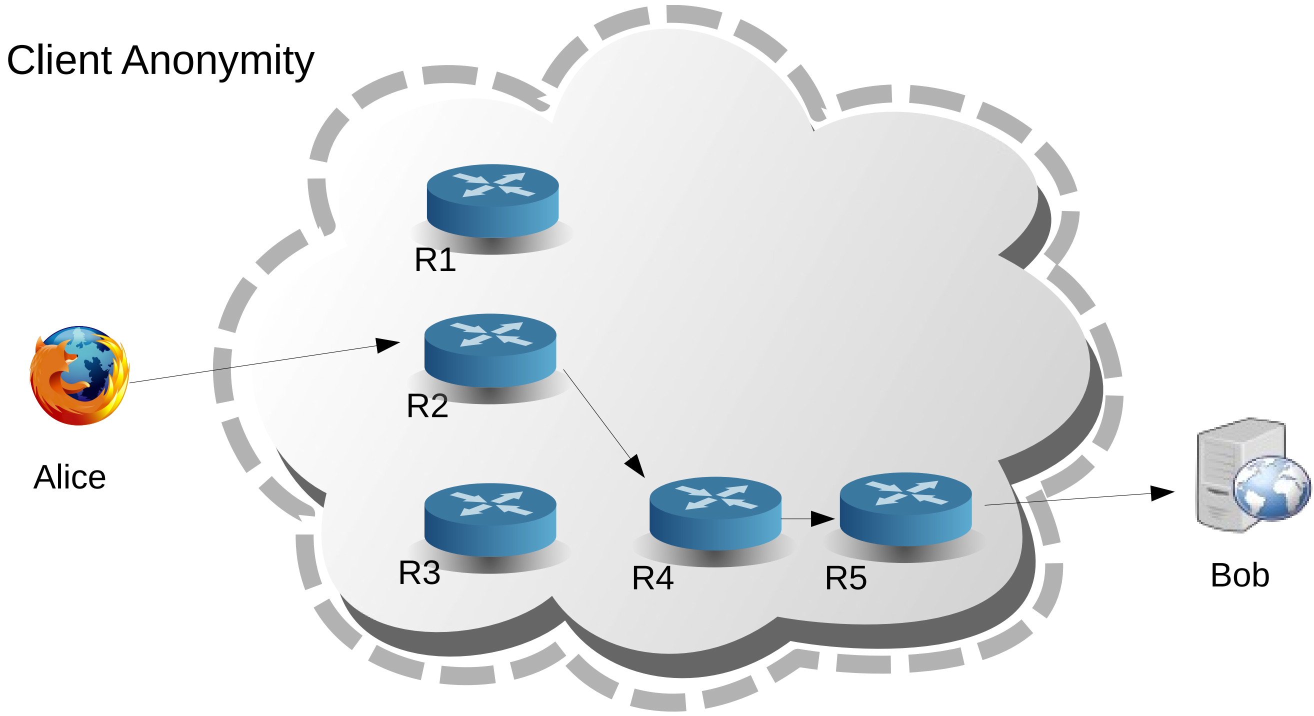
Paper Summary Tor is the most popular volunteer-based anonymity network consisting of over 3000 volunteer-operated relays. Apart from making connections to servers hard to trace to their origin it can also provide receiver privacy for Internet services through a feature called "hidden services". In this paper we expose flaws both in the design and implementation of Tor's hidden services that allow an attacker to measure the popularity of arbitrary hidden services, take down hidden services and deanonymize hidden services. We give a practical evaluation of our techniques by studying: (1) a recent case of a botnet using Tor hidden services for command and control channels; (2) Silk Road, a hidden service used to sell drugs and other contraband; (3) the hidden service of the DuckDuckGo search engine.

Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization

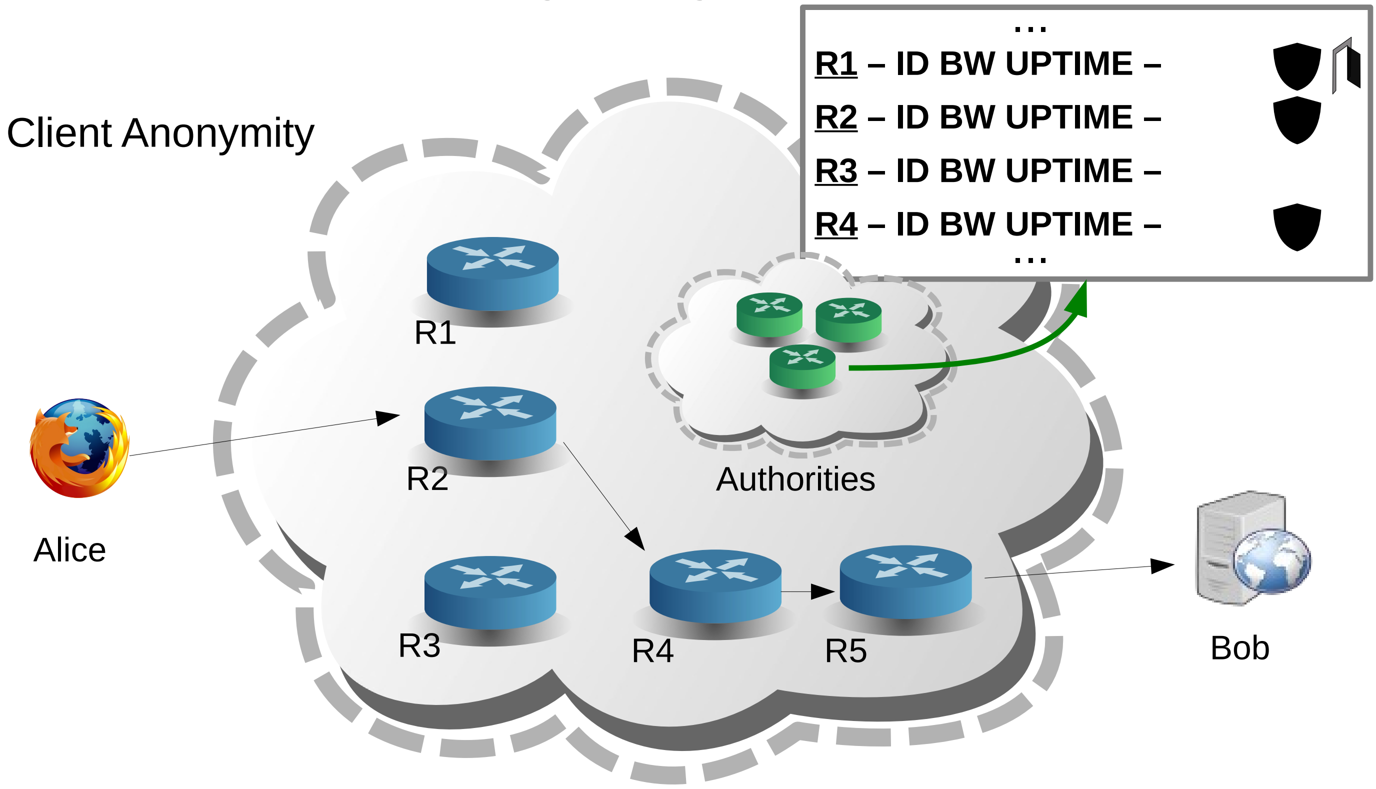
A. Biryukov, I. Pustogarov, R.P. Weinmann
University of Luxembourg
Ivan.pustogarov@uni.lu

May 20, 2013

Tor anonymity network



Tor anonymity network

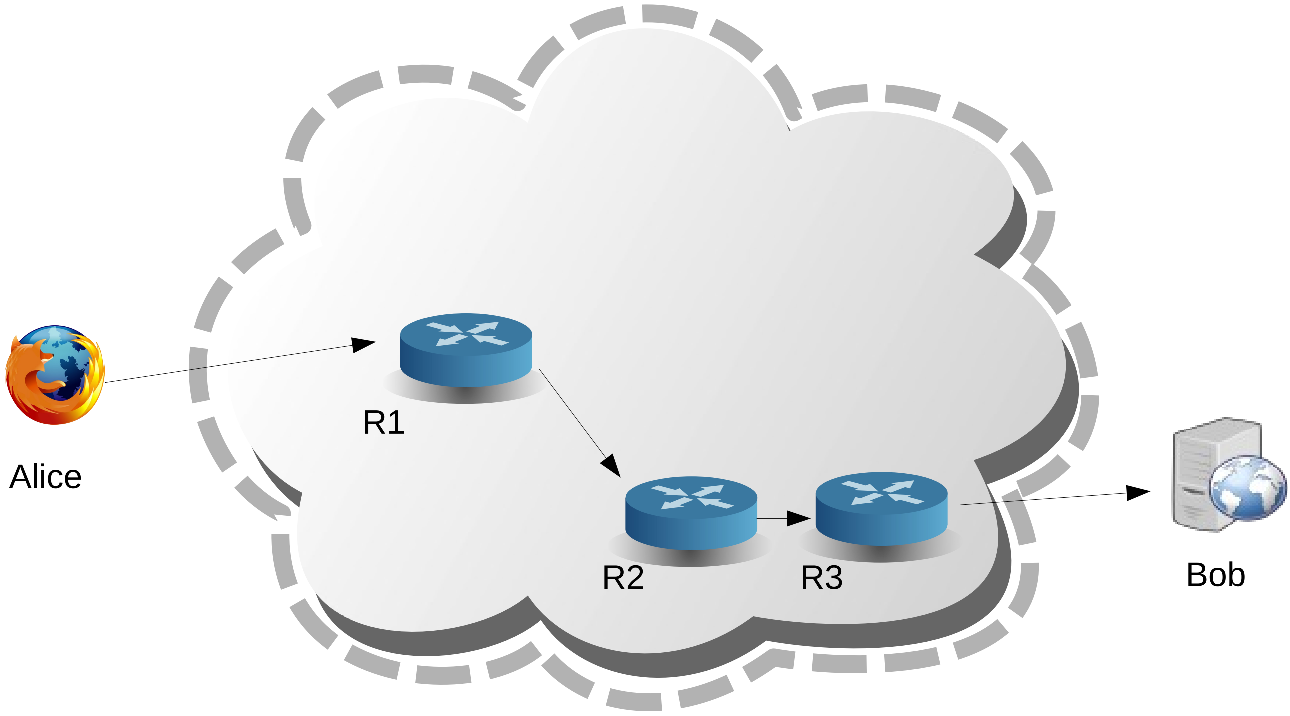


Consensus

menTor	1737	67 d	55863896.cust.multi.fi [85.134.56.150]
microshaft	2820	66 d	tor-exit.microshaft.org [208.201.249.3]
minisausage	3348	35 d	50.7.184.58 [50.7.184.58]
morphium	298	51 d	this.is.a.Tor.server.please.see.tor.morphium.info [91.143.90.25]
NetromAc	2115	47 d	1385160742.business.dbnet.dk [82.143.224.38]
Nitr0x	175	78 d	50.97.1.36-static.reverse.softlayer.com [50.97.1.36]
OhCanada	419	51 d	van1.zworg.com [209.17.191.117]
onconnex80	392	213 d	tor01.onconnex.com [184.105.231.11]
PasToutAFaitNet1	261	176 d	91.229.20.159 [91.229.20.159]
PasToutAFaitNet2	763	196 d	tor2.pastoutafait.net [95.130.11.247]
plebia	3599	79 d	tor-exit.plebia.org [37.59.162.218]
pps	9	59 d	184-22-164-107.static.hostnoc.net [184.22.164.107]
PrivaTOReu	4229	100 d	torexit.privator.eu [88.208.90.1]
programmercpp	149	36 d	proxy [213.171.220.40]
PsyNetNP	155	52 d	broadband-95-84-148-164.nationalcablenetworks.ru [95.84.148.164]
Qwerty	91	157 d	93.167.245.178 [93.167.245.178]

http://torstatus.blutmagie.de/

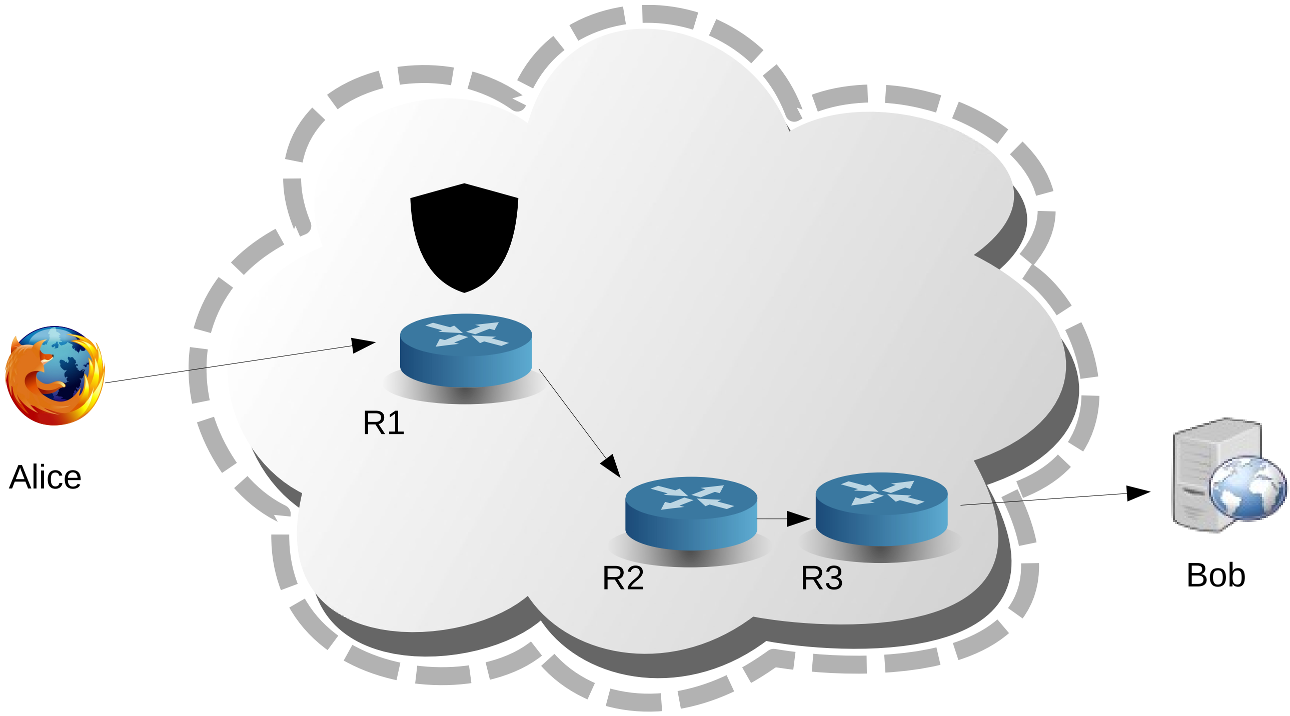
Guards



Guard = high uptime + high bandwidth

Every client has 3 Guard nodes

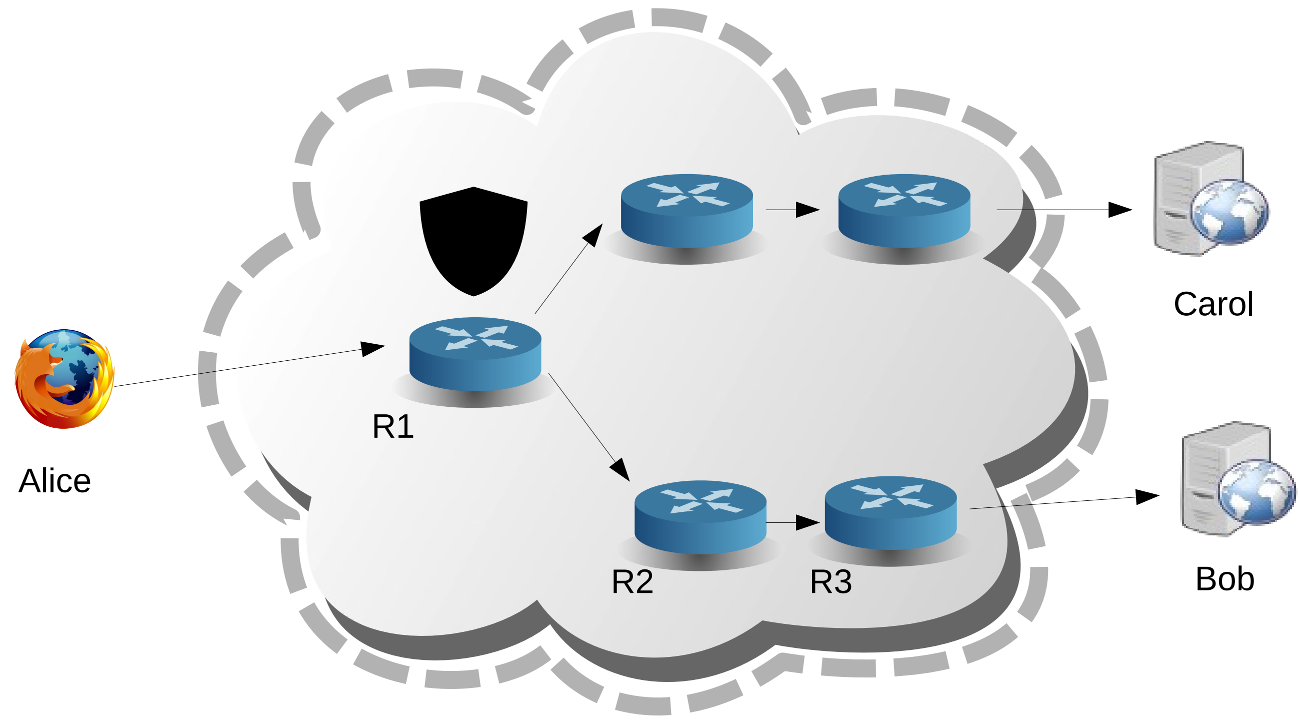
Guards



Guard = high uptime + high bandwidth

Every client has 3 Guard nodes

Guards

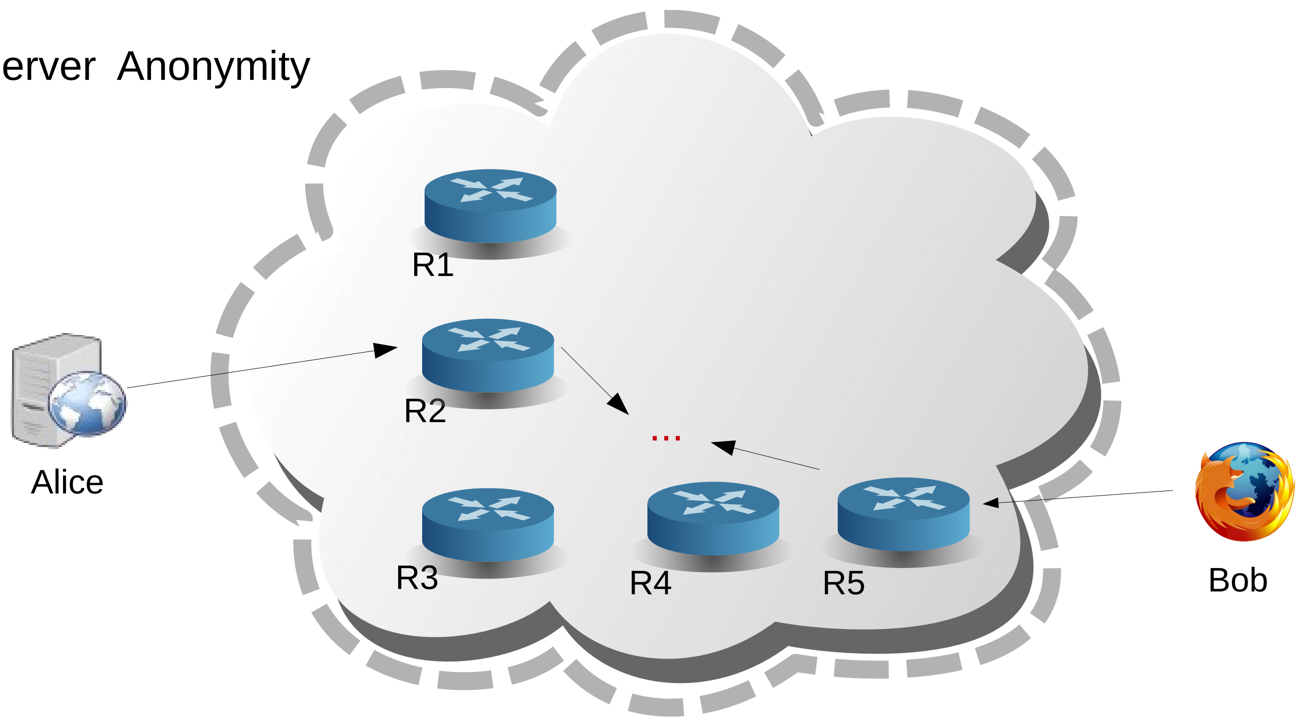


Guard = high uptime + high bandwidth

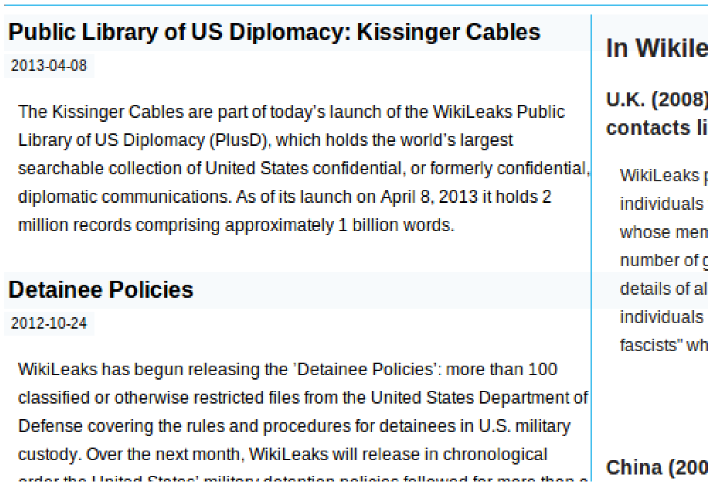
Every client has 3 Guard nodes

.onion

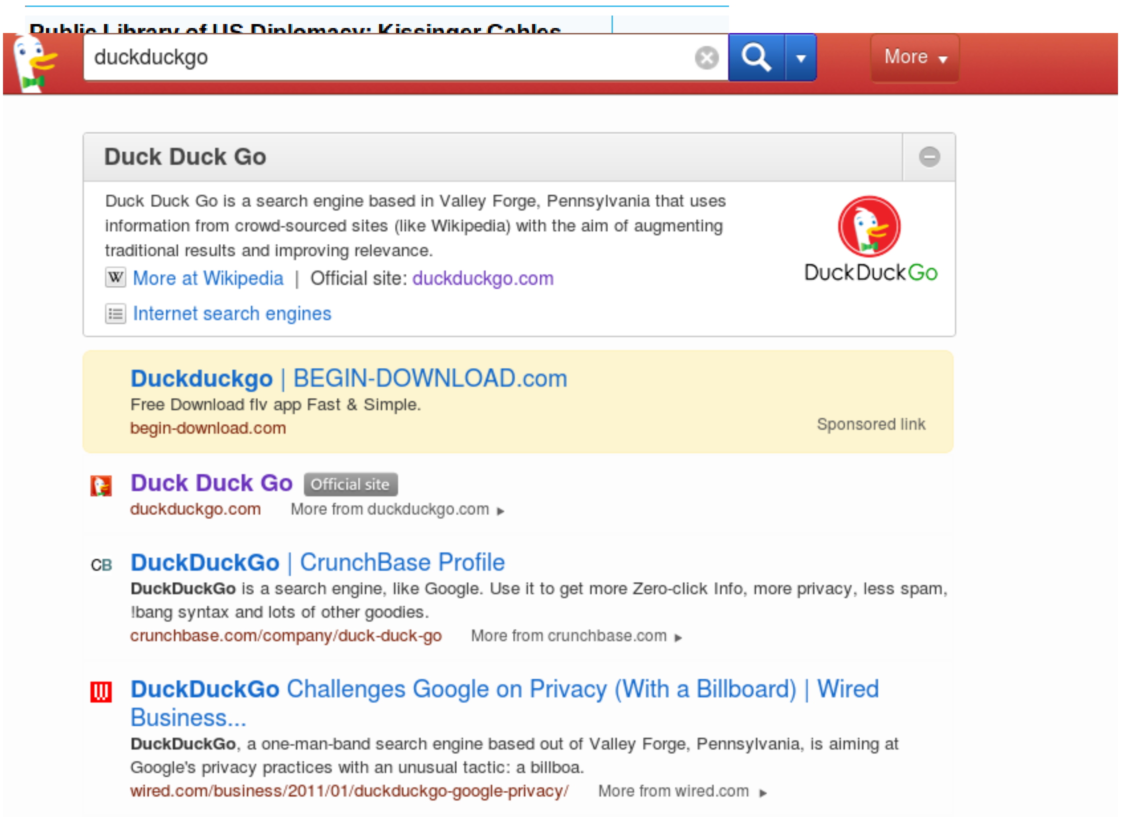
Server Anonymity



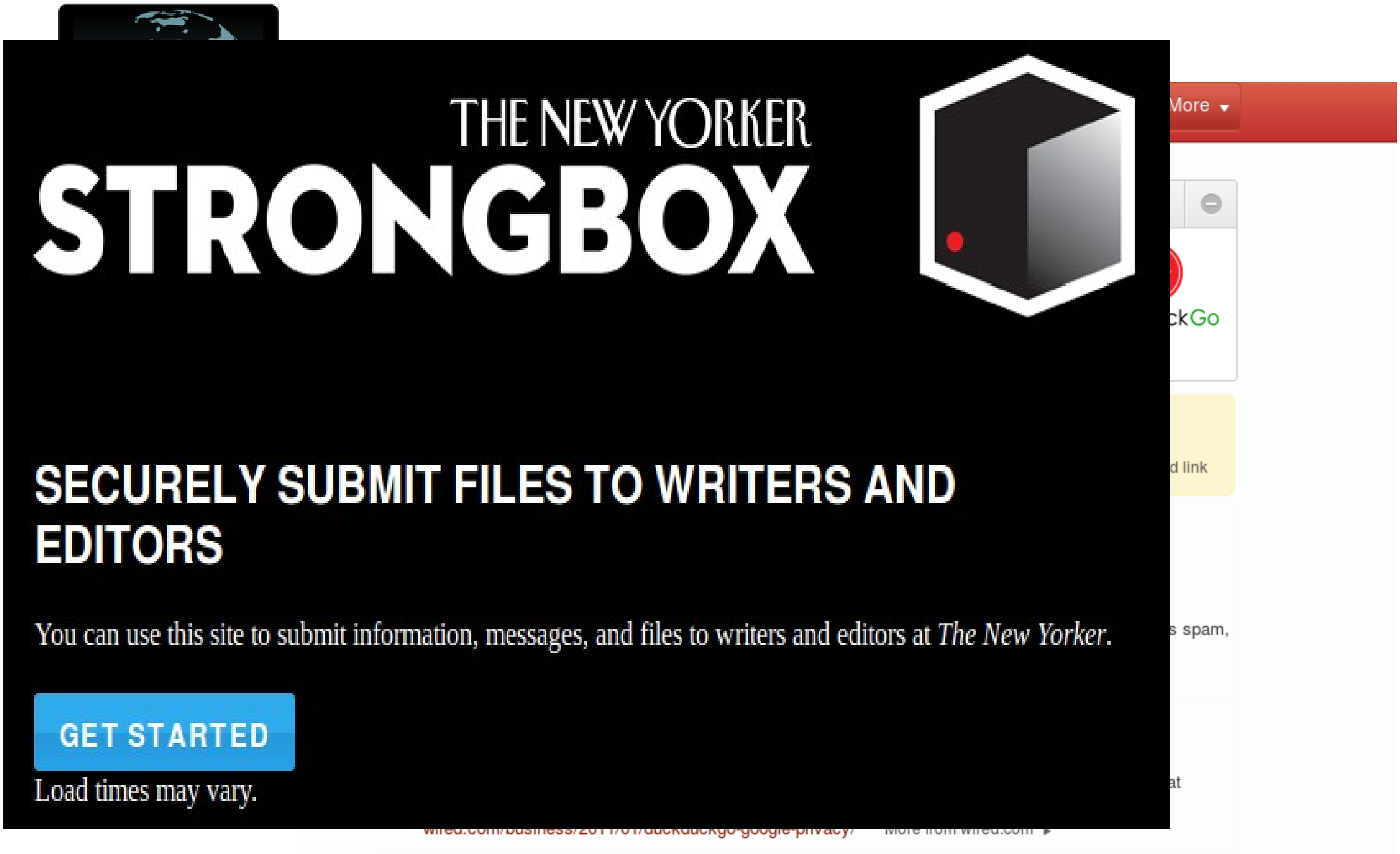
Examples of Tor HS



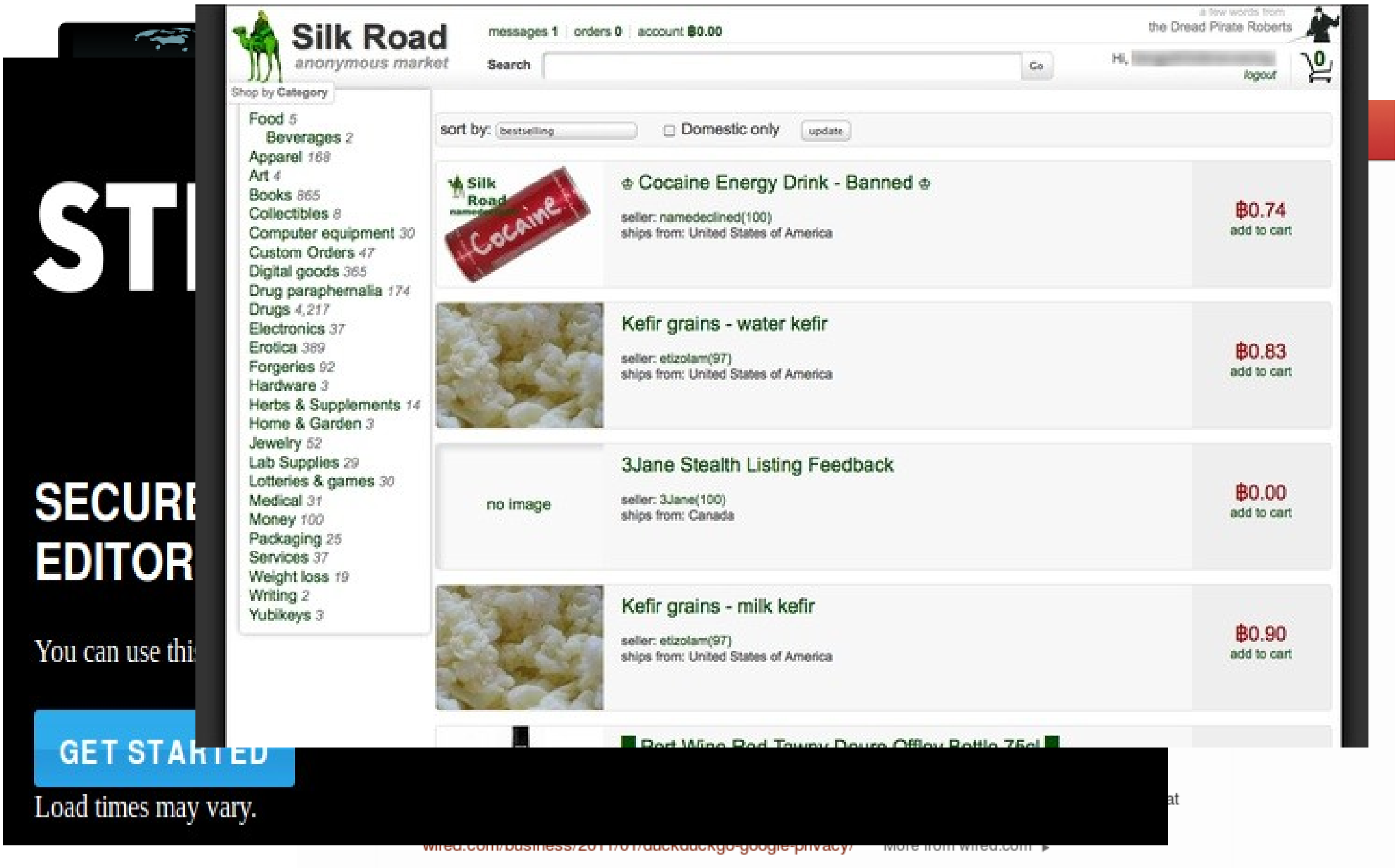
Examples of Tor HS



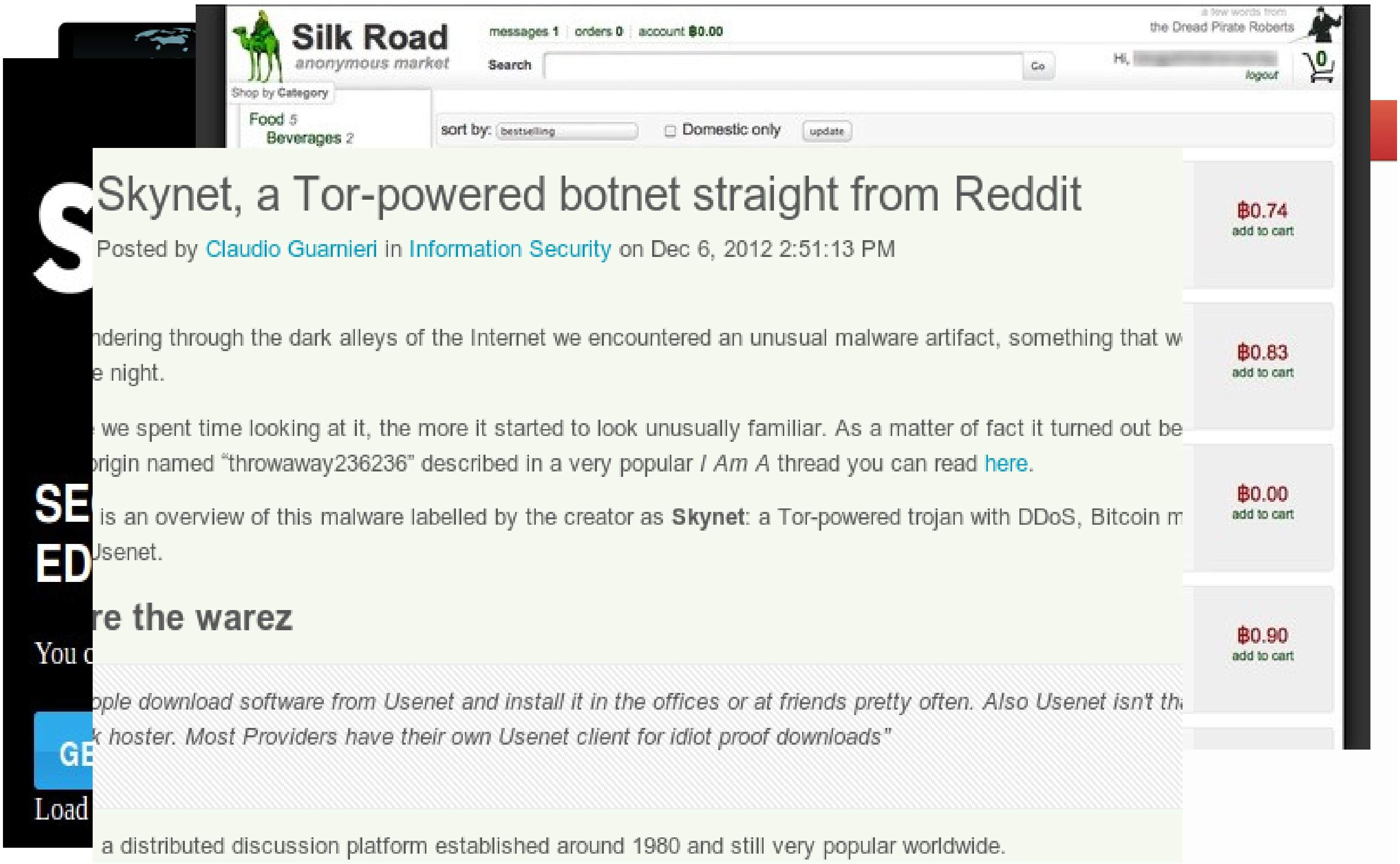
Examples of Tor HS



Examples of Tor HS



Examples of Tor HS

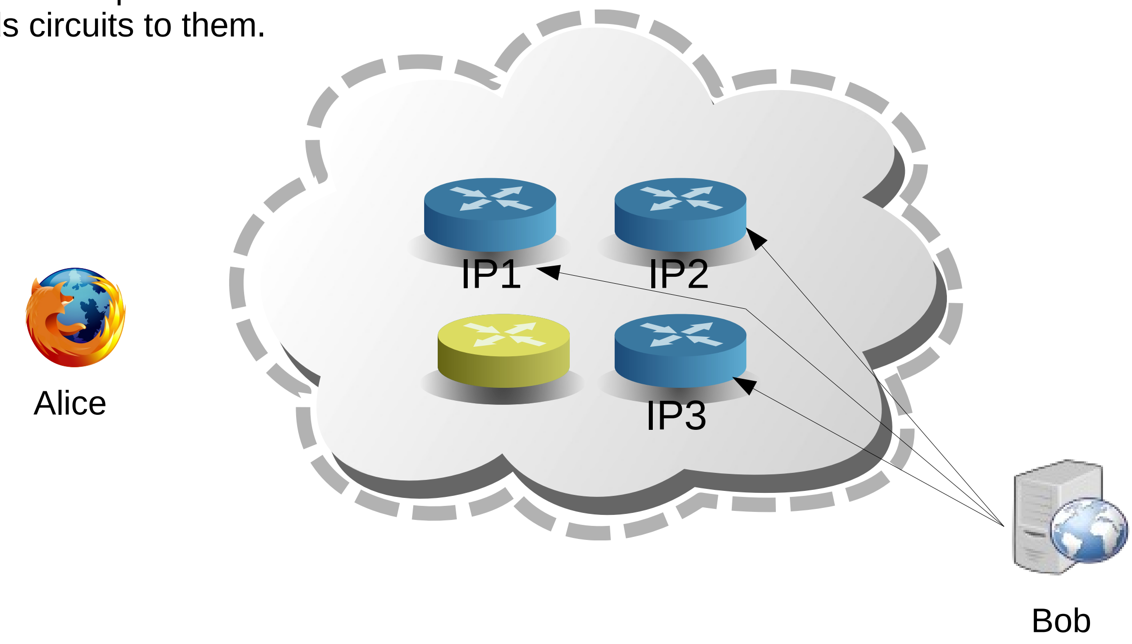


.onion security

Tracking Popularity	
Denial of Service	
Collecting onion addresses	
Revealing Guard Nodes	
Deanonymisation	

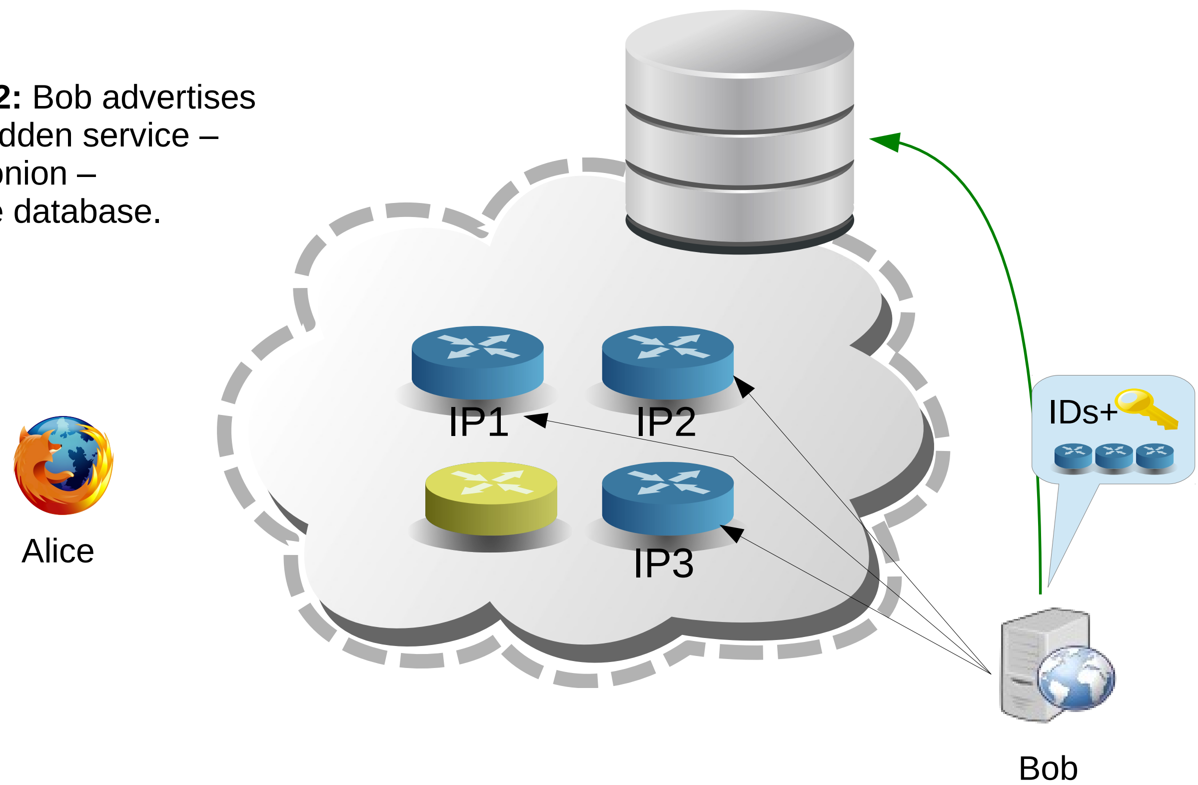
Tor rendezvous protocol

Step1: Bob picks some introduction points and builds circuits to them.



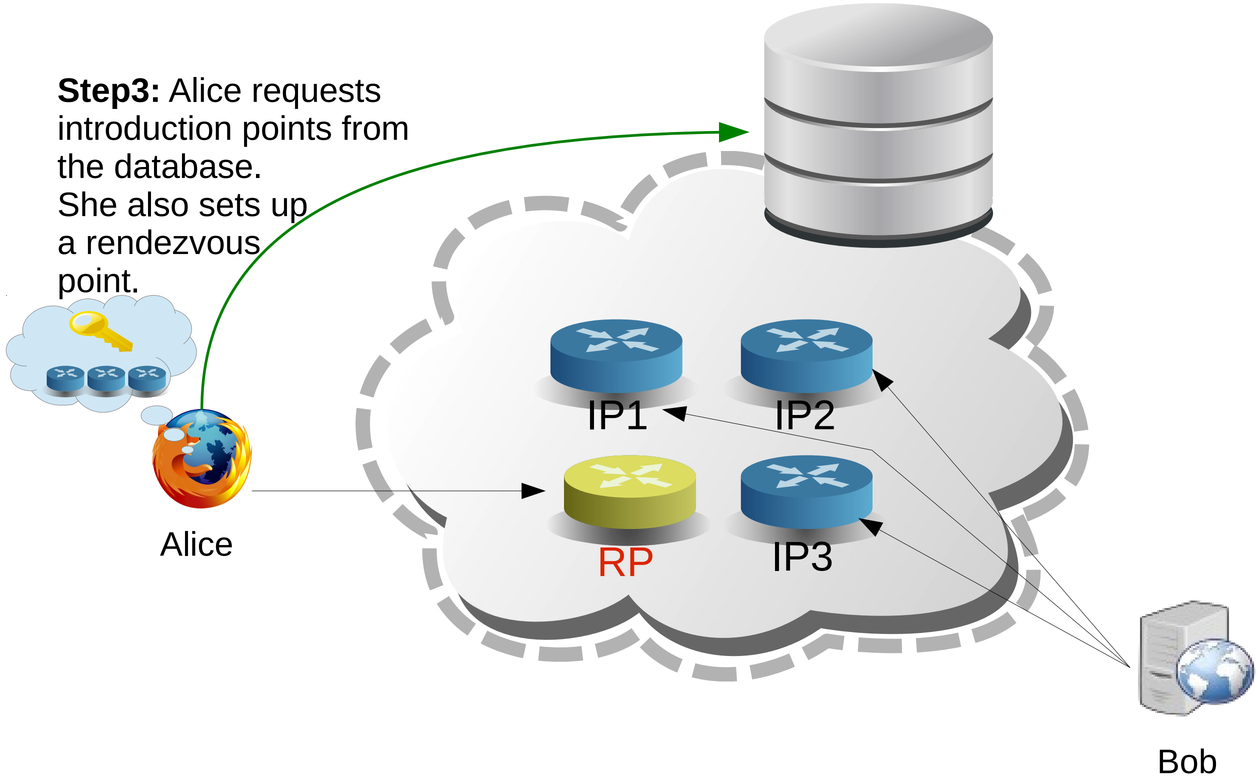
Tor rendezvous protocol

Step2: Bob advertises his hidden service – <z>.onion – at the database.



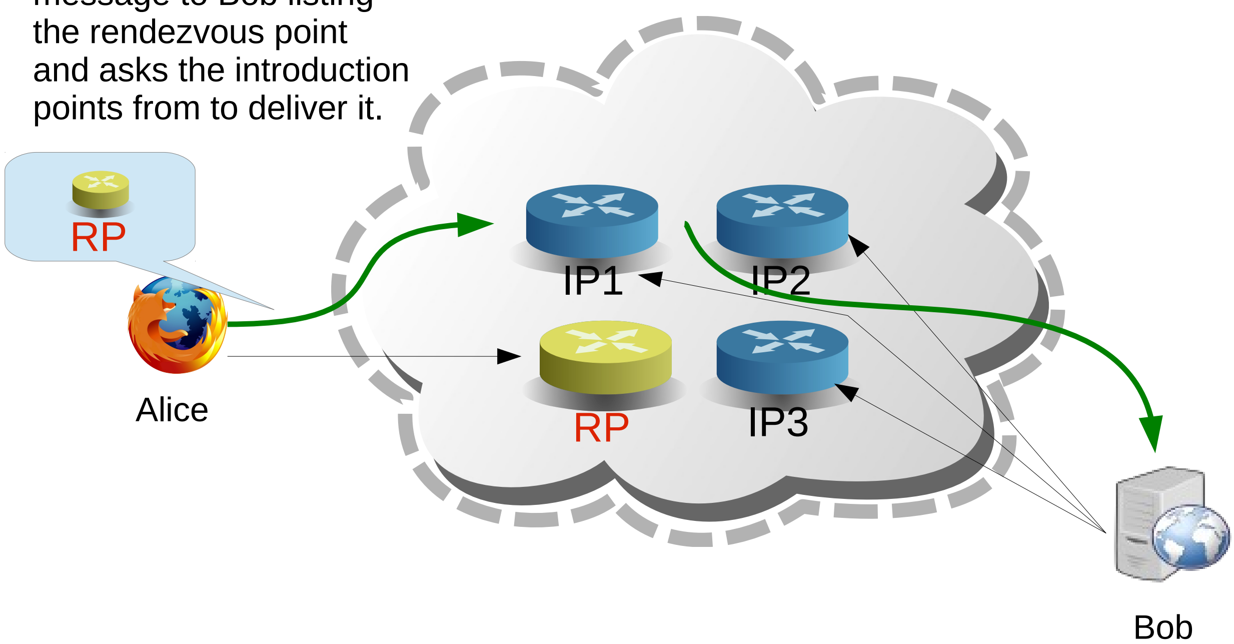
Tor rendezvous protocol

Step3: Alice requests introduction points from the database.
She also sets up a rendezvous point.



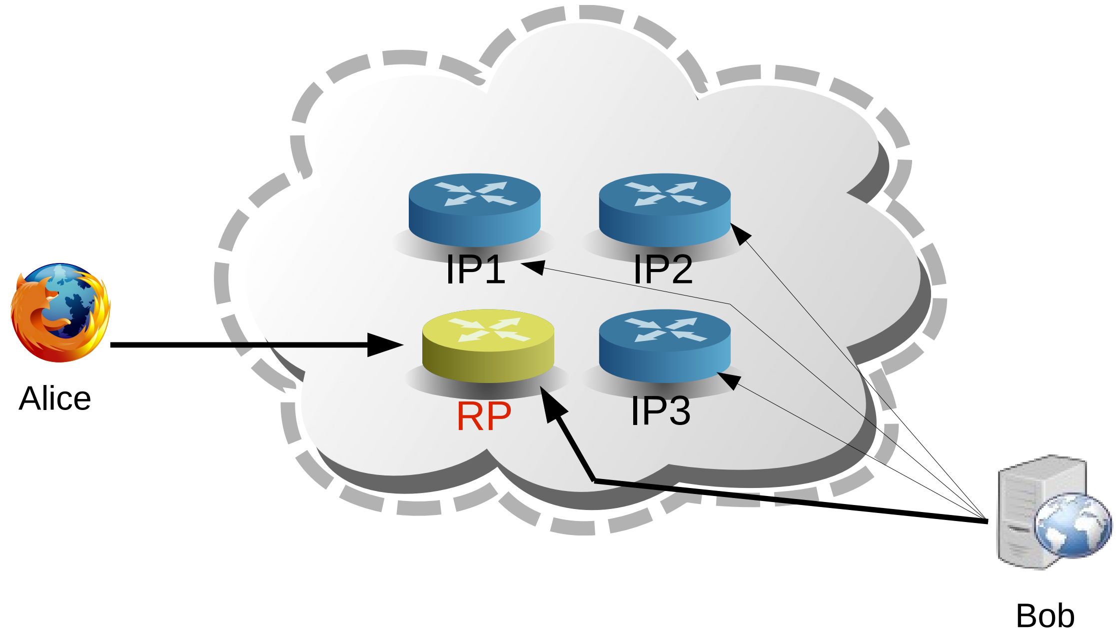
Tor rendezvous protocol

Step4: Alice sends a message to Bob listing the rendezvous point and asks the introduction points from to deliver it.

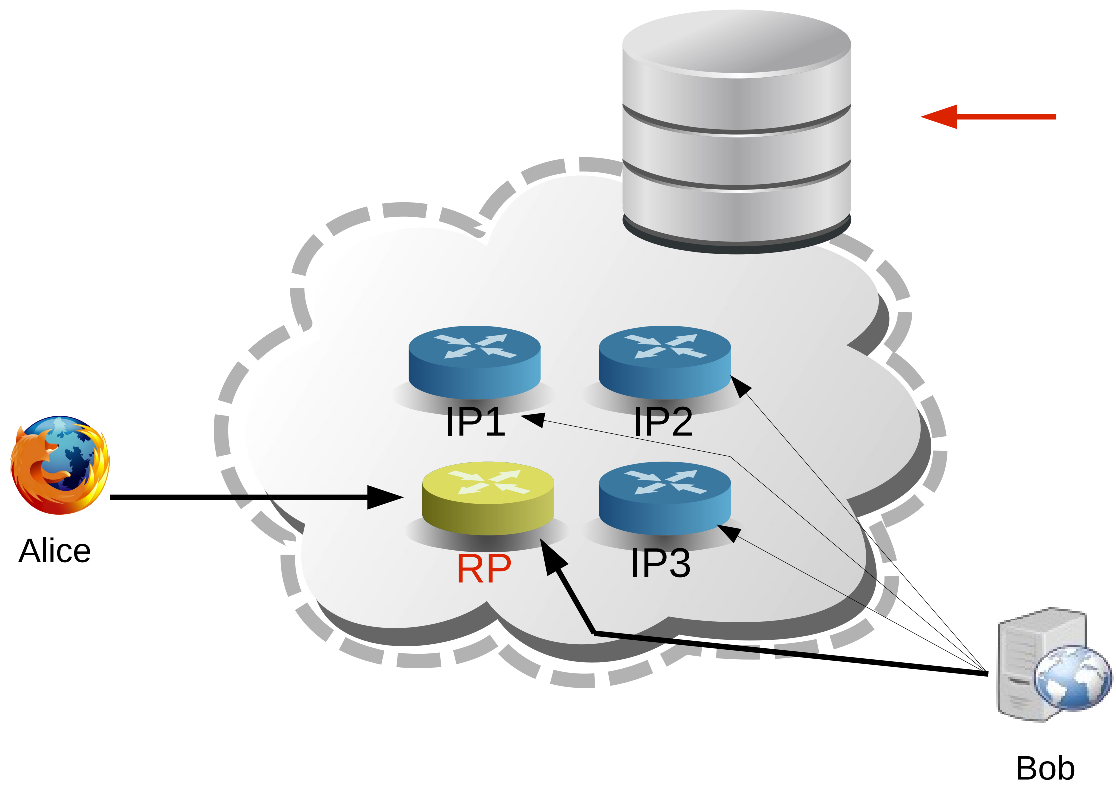


Tor rendezvous protocol

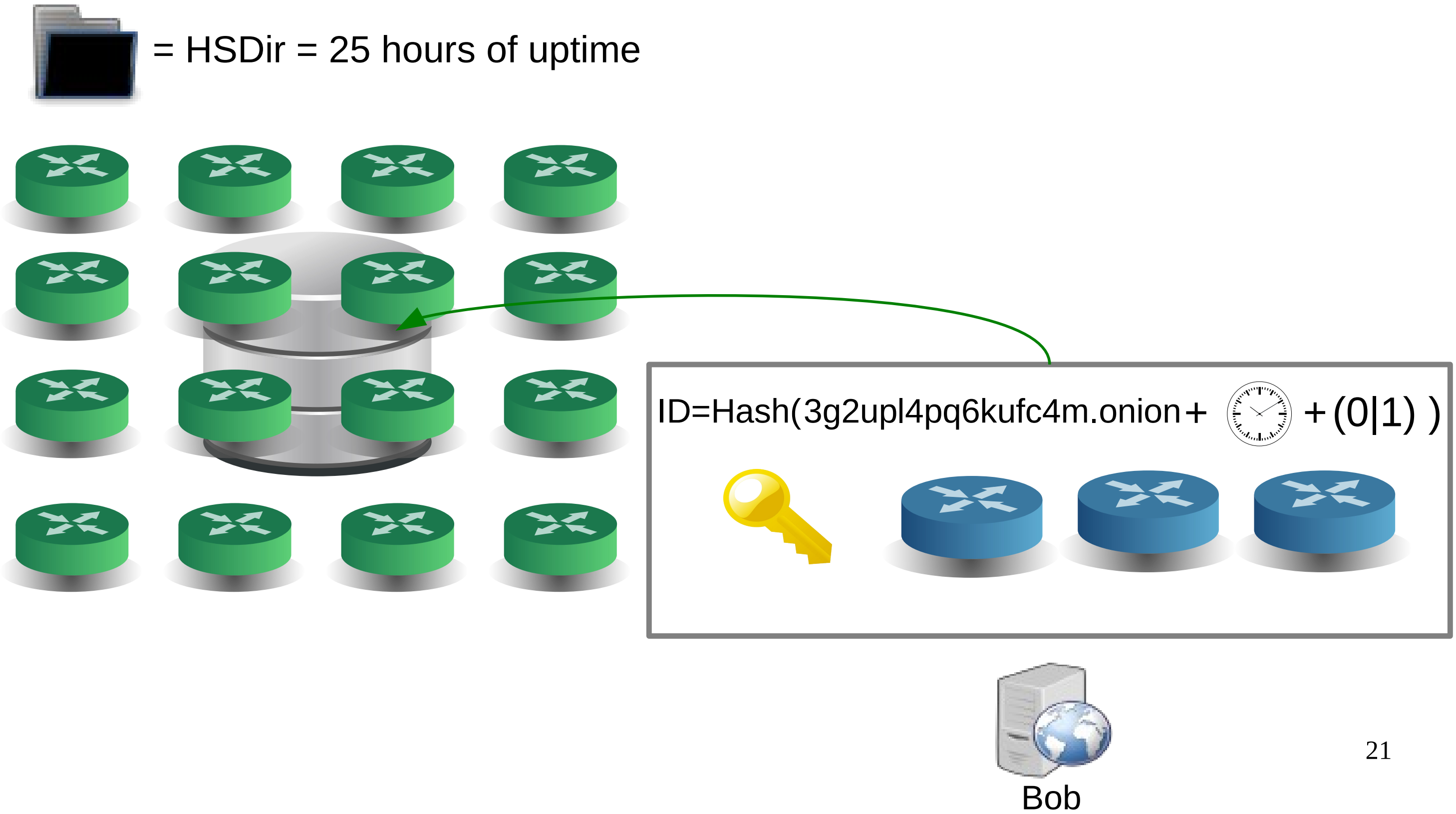
Step5: Alice and Bob
Connect at the Rendezvous
point



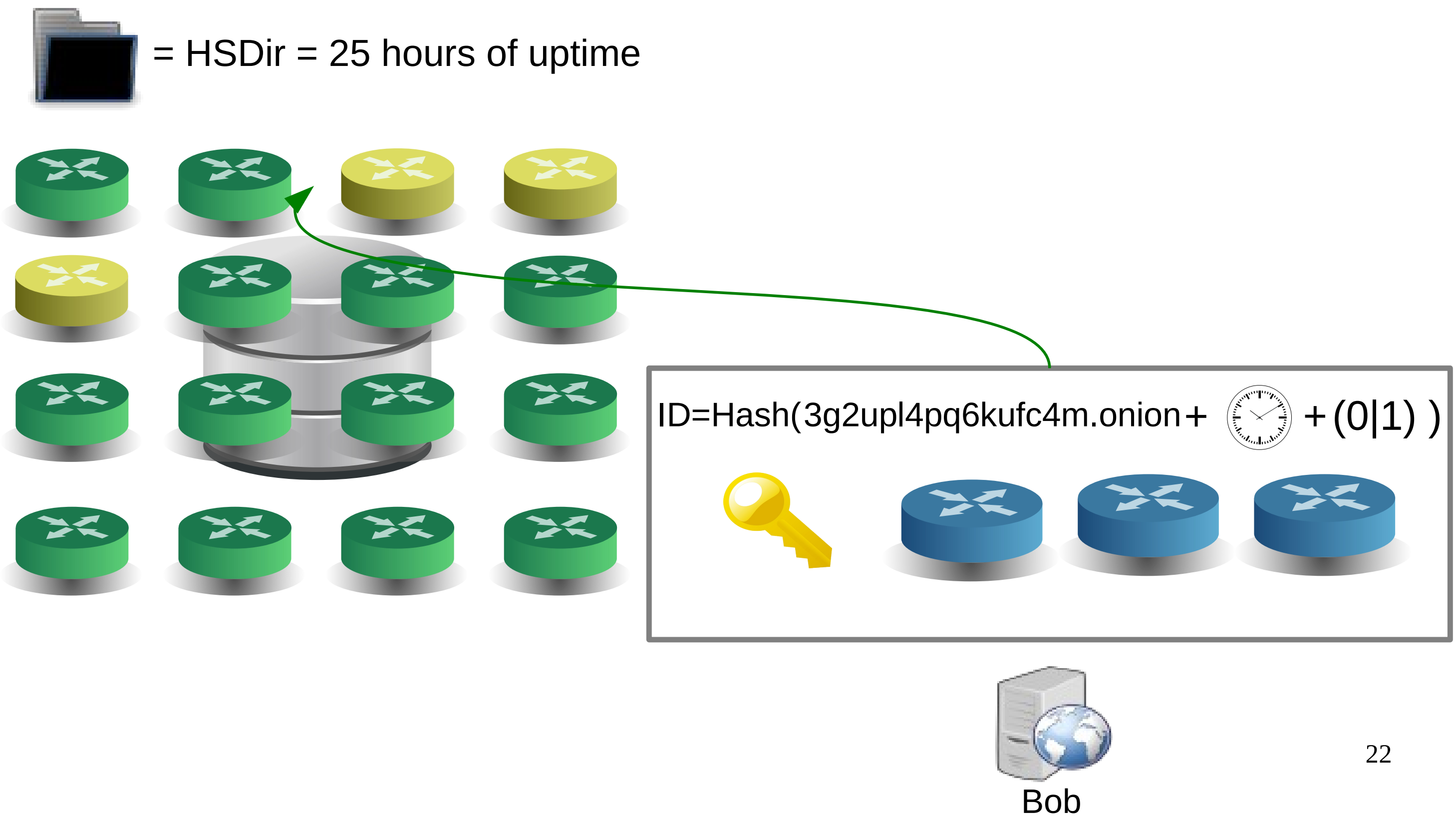
Tor rendezvous protocol



Responsible hidden service directories



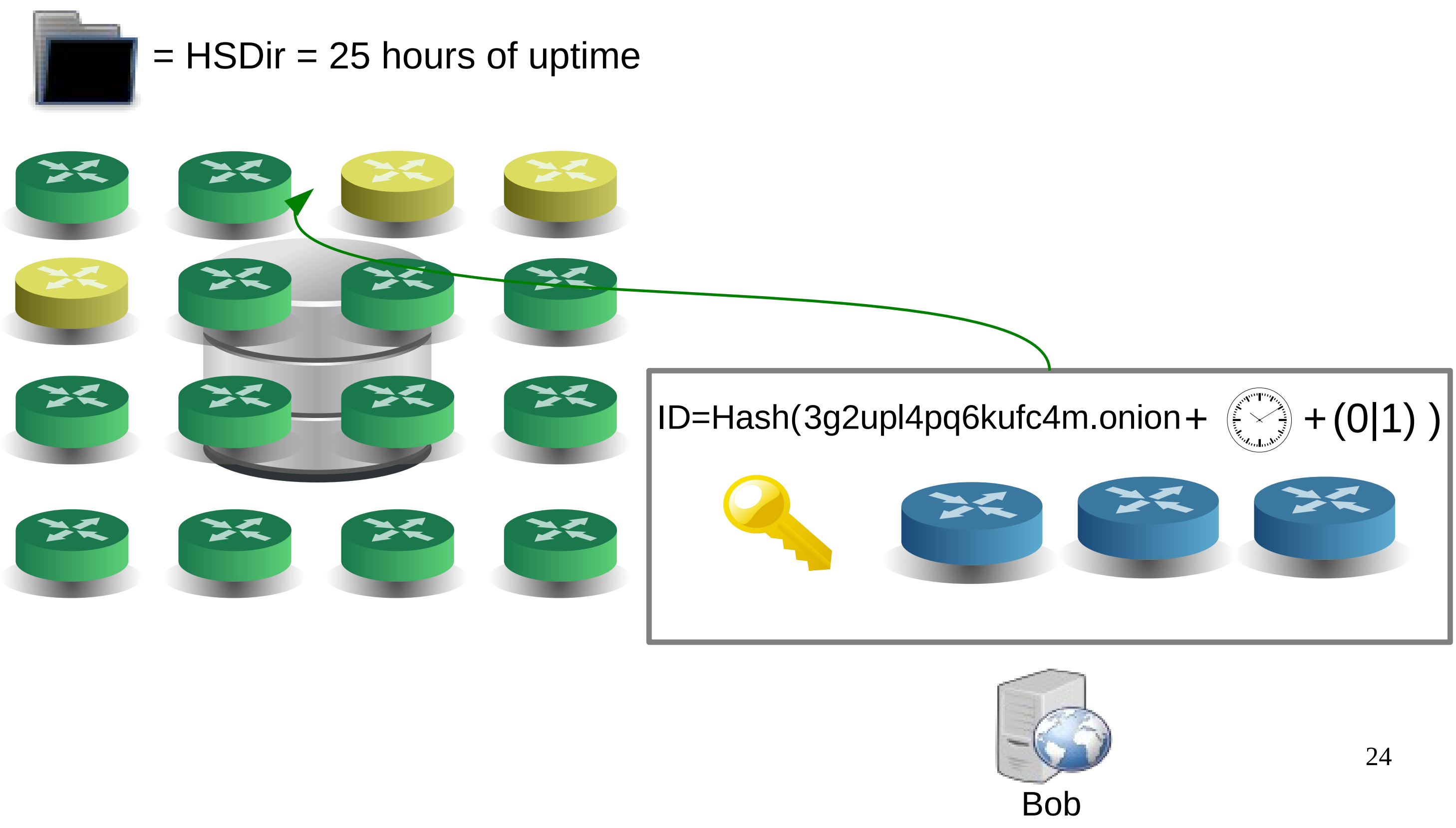
Responsible hidden service directories



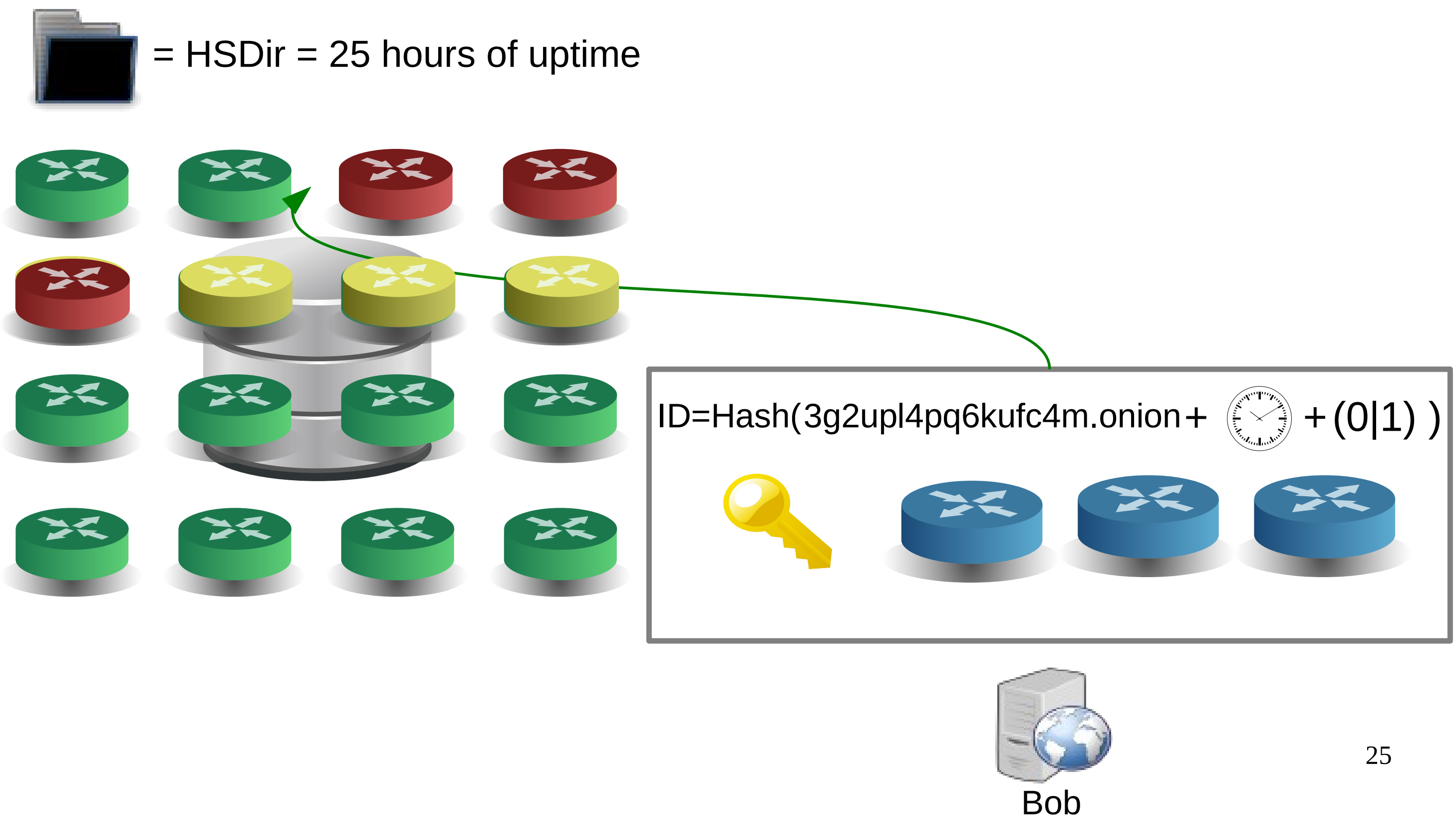
Outline

Tracking Popularity	
Denial of Service	
Collecting onion addresses	
Revealing Guard Nodes	
Deanonymisation	

Responsible hidden service directories

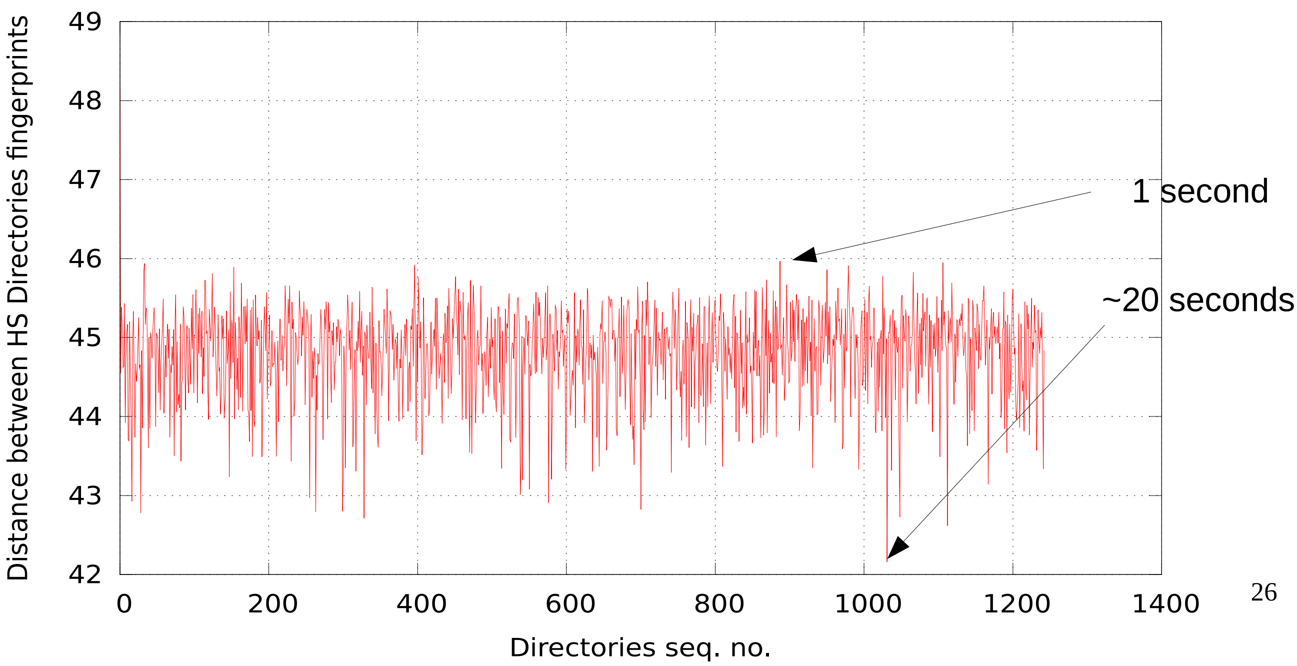


Responsible hidden service directories



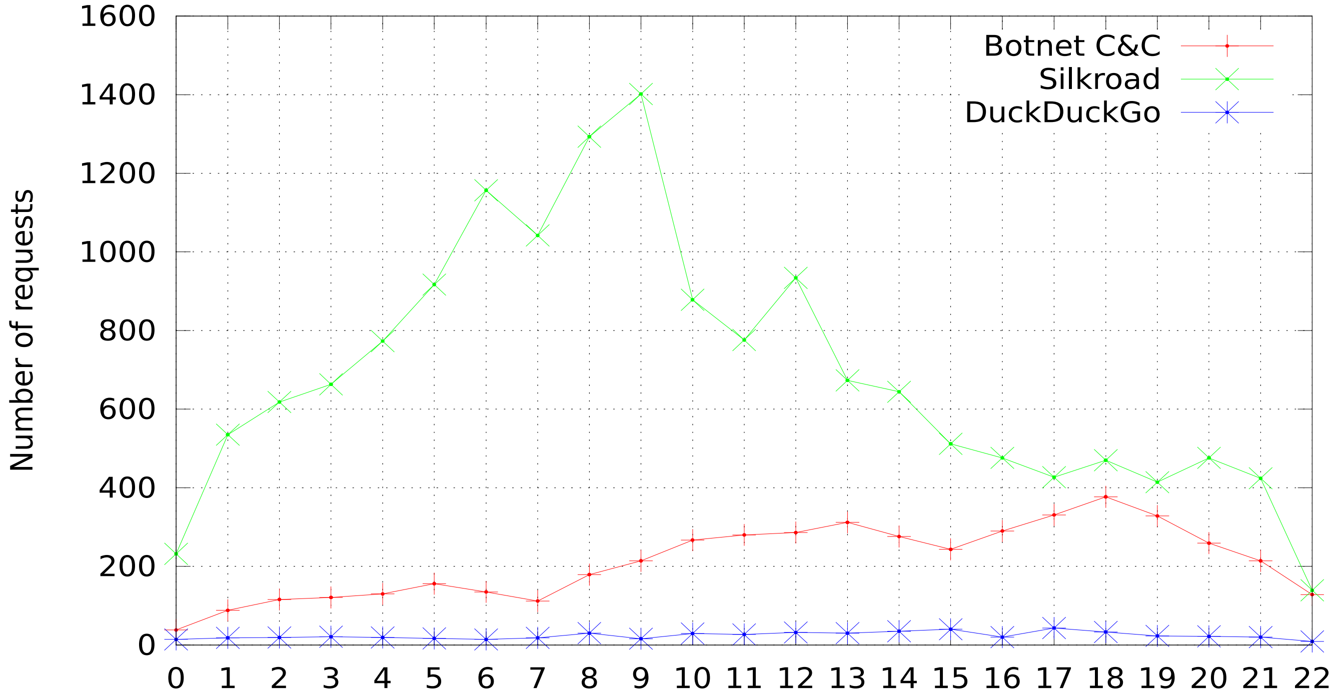
Impersonating Hidden service directory

- By impersonating 1 directory, we can track the popularity
- By impersonating all 6 directories, we can DoS.



Tracking popularity

- We tracked popularity of Skynet C&C, Silkroad, and DuckDuckGo



Outline

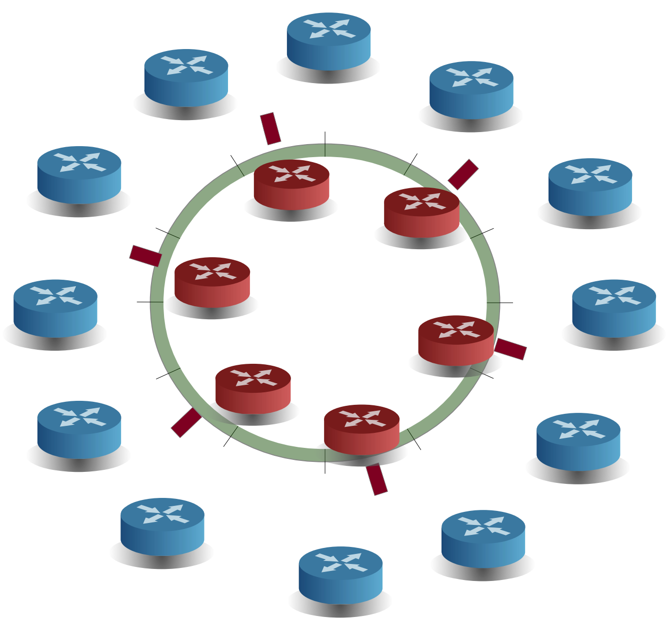
Tracking Popularity	
Denial of Service	
<u>Collecting onion addresses</u>	
Revealing Guard Nodes	
Deanonymisation	

.onion harvesting

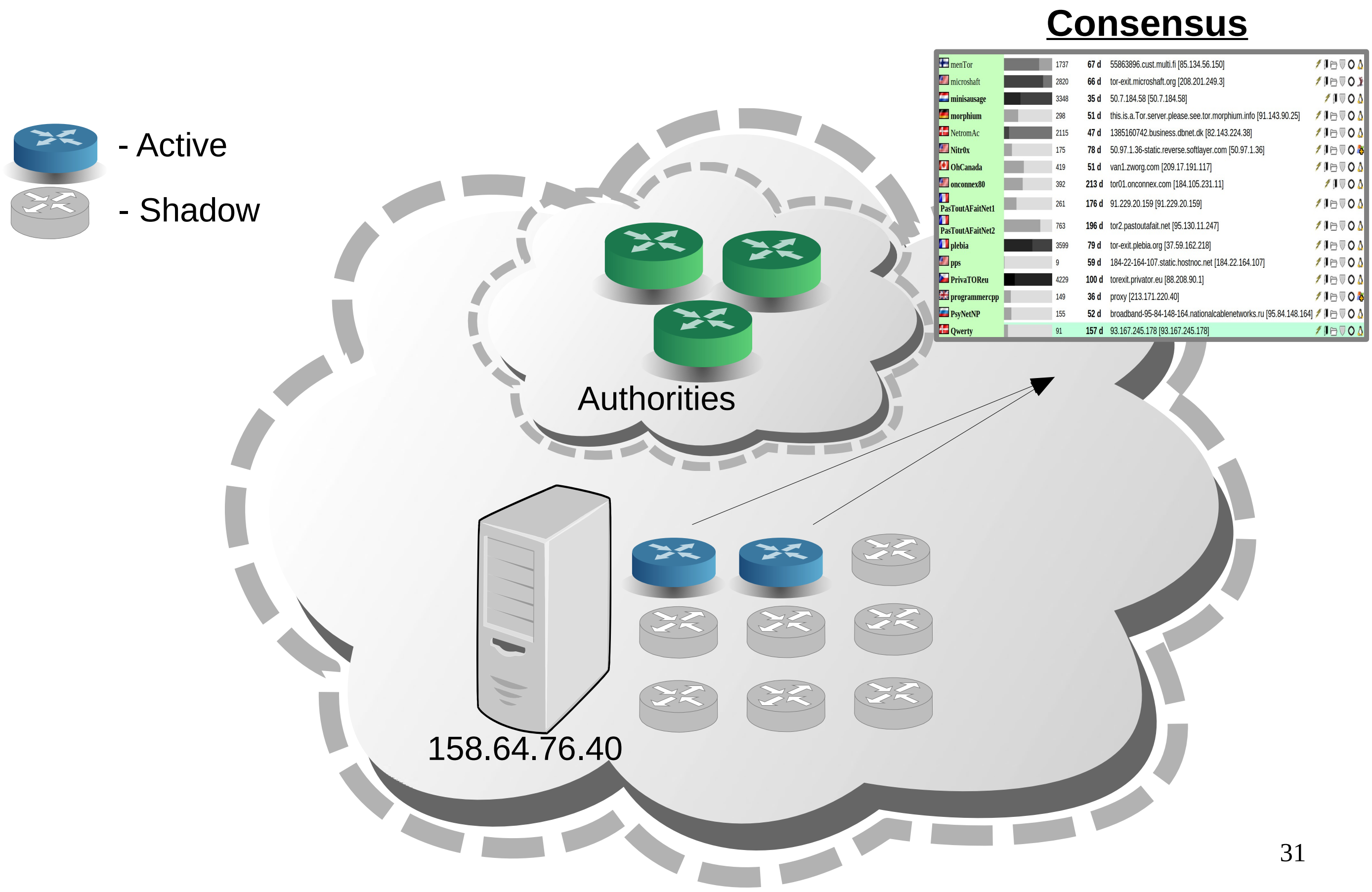
- Problems
  - Distributed storage
  - Cannot query HSDirs
  - No links between different .onion addresses => cannot use traditional crawling

Collecting onion addresses

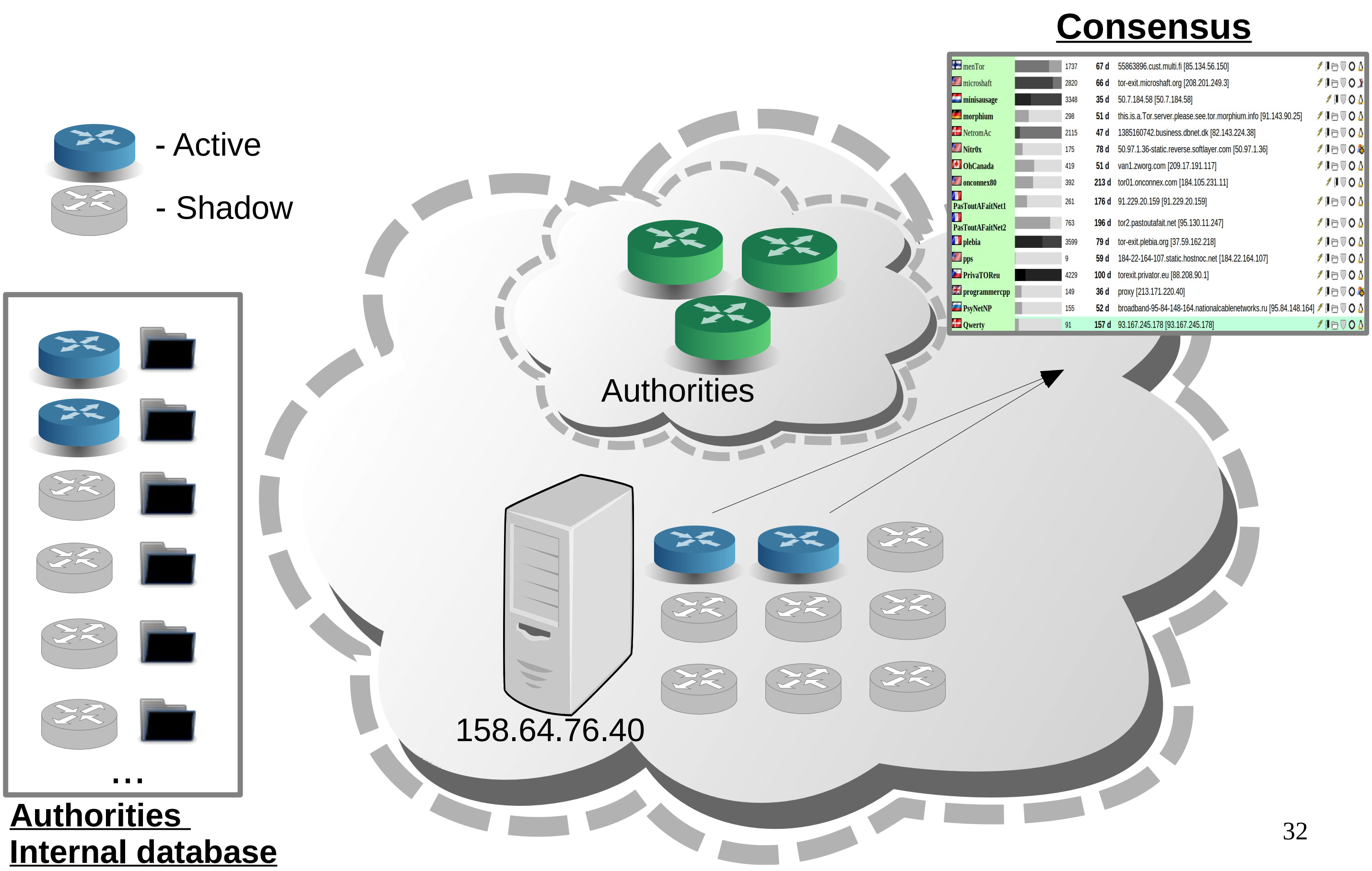
- Naive approach will require ~350 IP addresses.



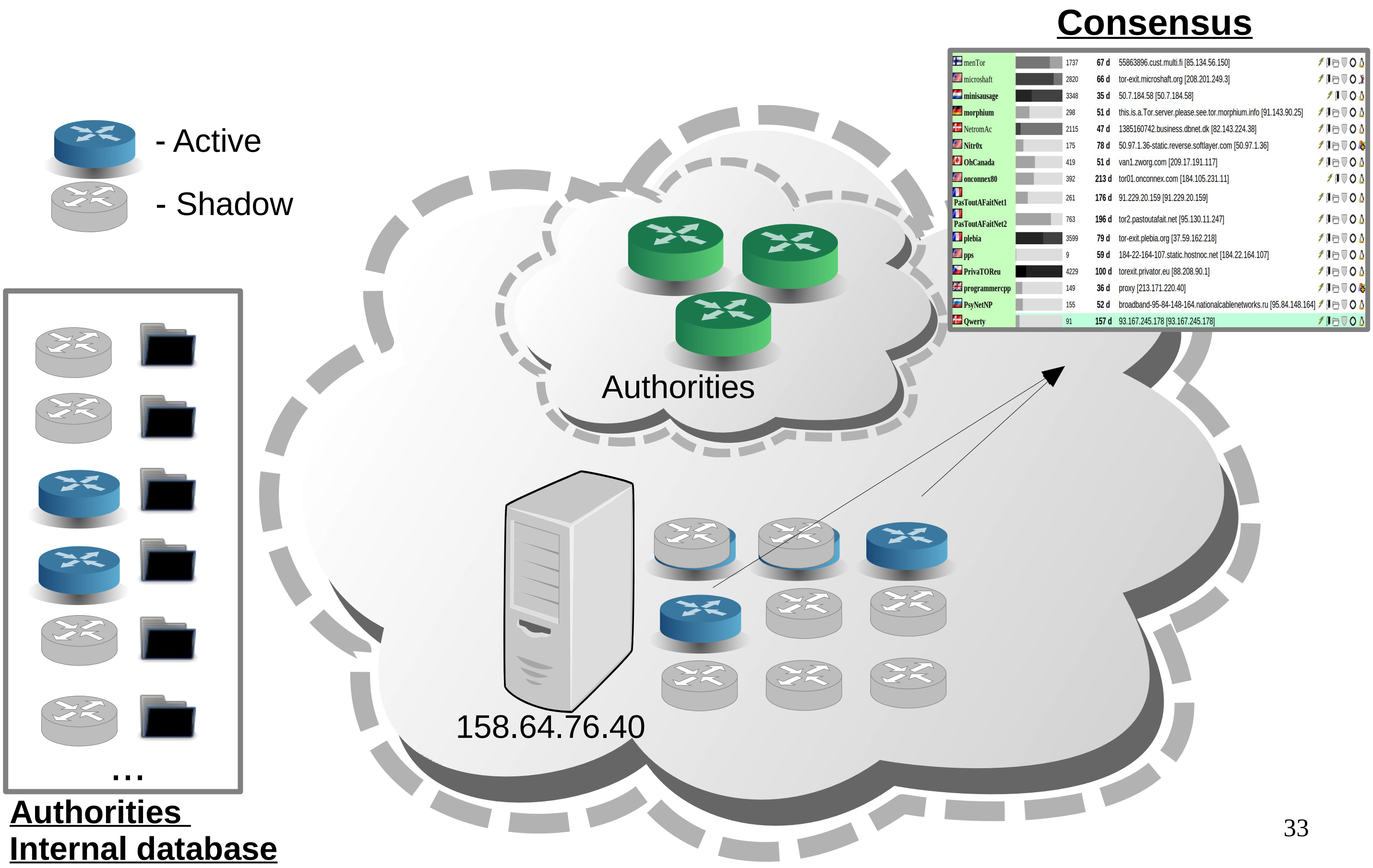
Shadowing



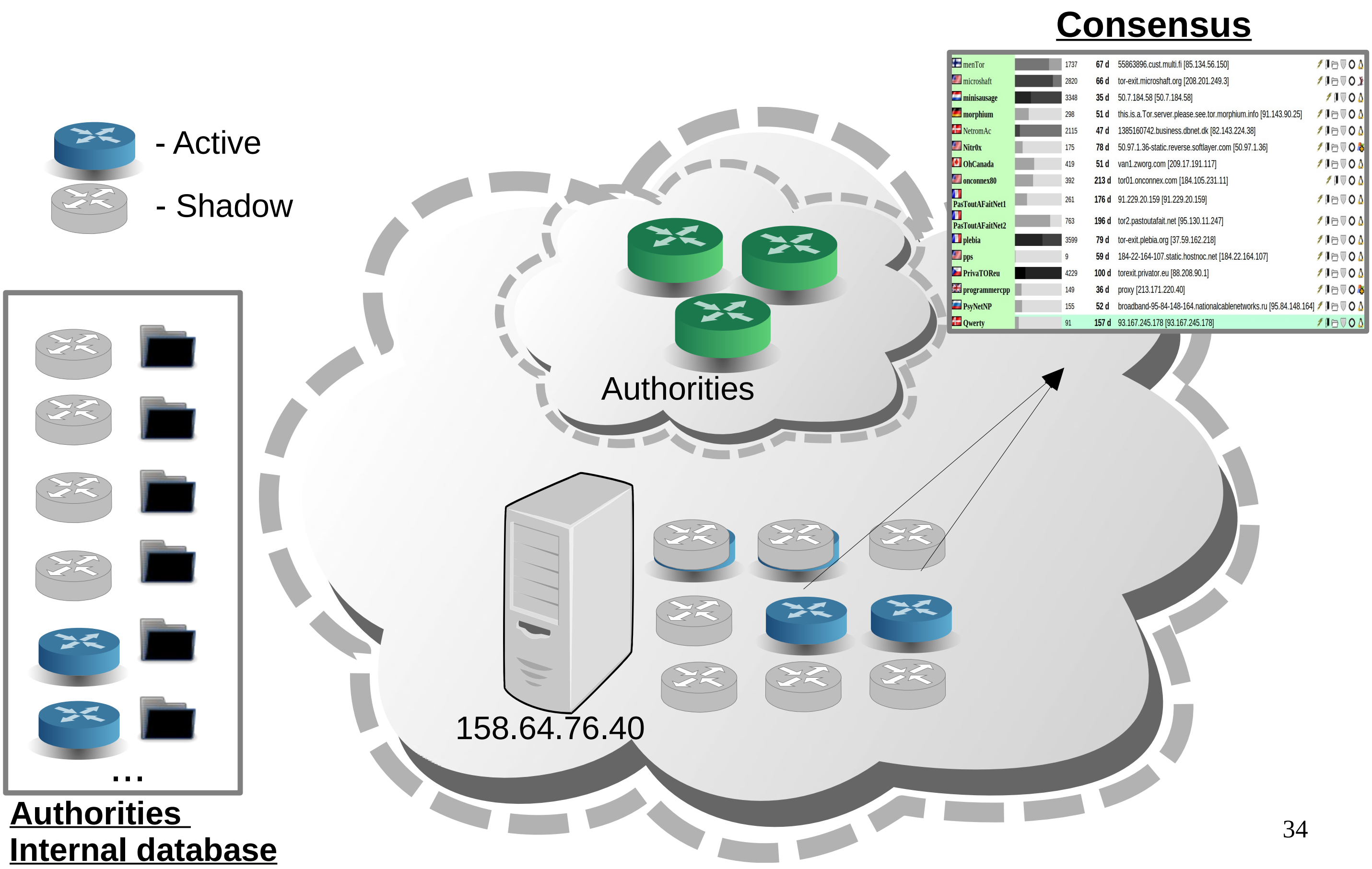
Shadowing



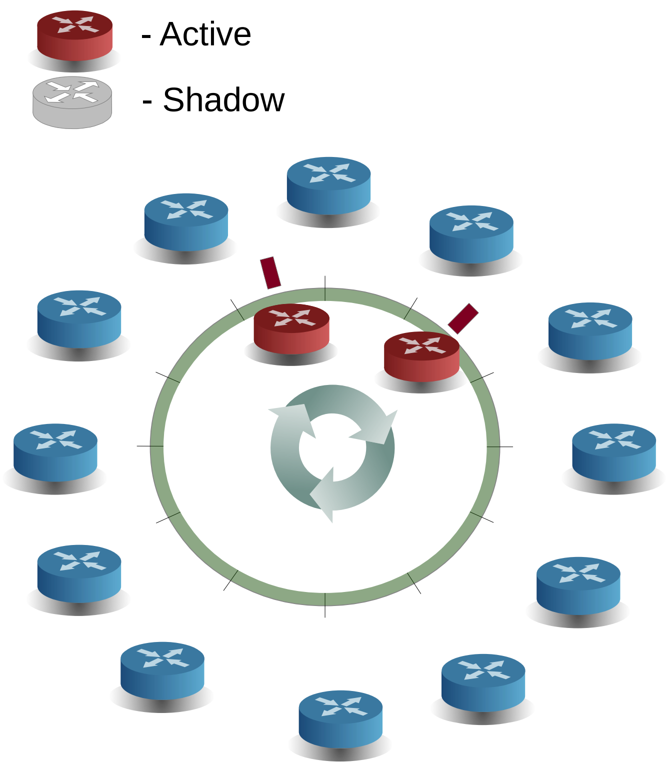
Shadowing



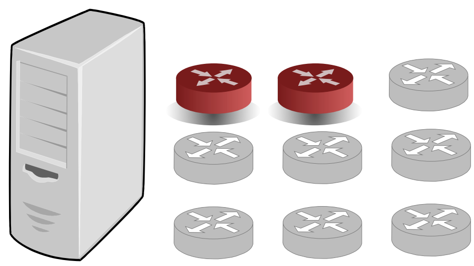
Shadowing



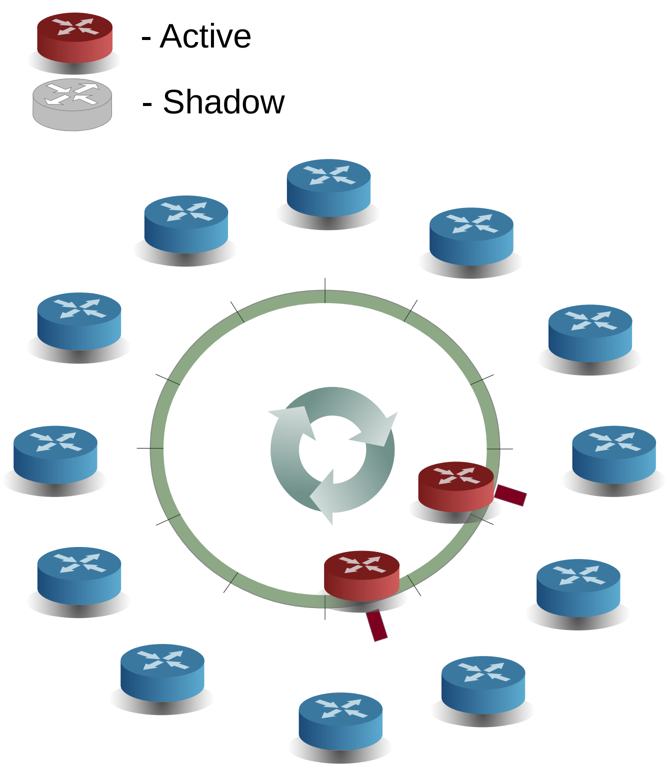
Collecting onion addresses



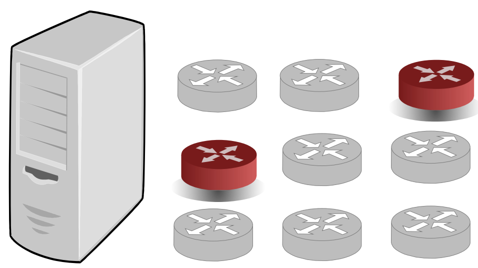
- Naive approach will require ~350 IP addresses.
- Descriptors don't relocate within 24 hours.
- Prepare shadow HSDir relays and gradually pull to consensus.



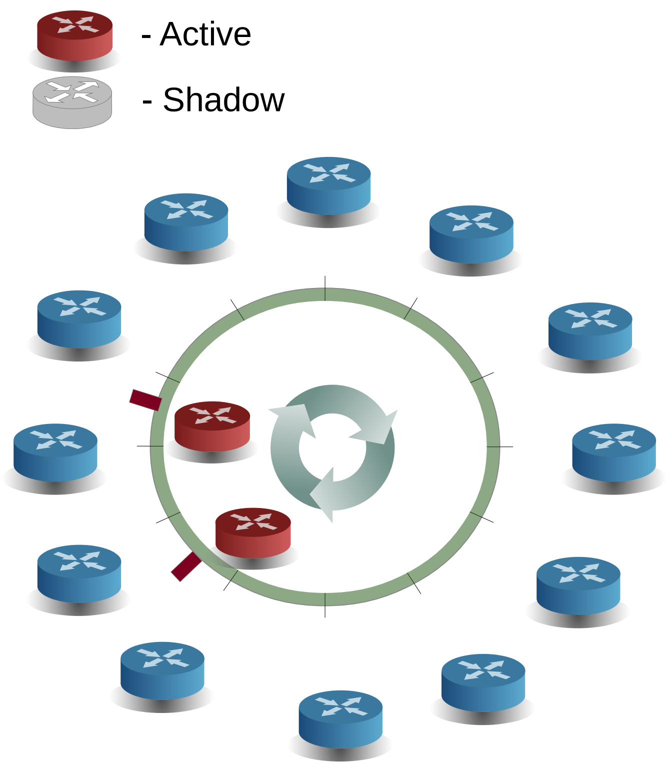
Collecting onion addresses



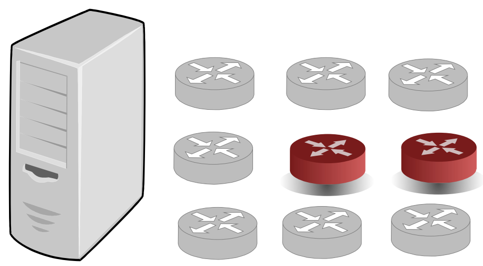
- Naive approach will require ~350 IP addresses.
- Descriptors don't relocate within 24 hours.
- Prepare shadow HSDir relays and gradually pull to consensus.



Collecting onion addresses



- Naive approach will require ~350 IP addresses.
- Descriptors don't relocate within 24 hours.
- Prepare shadow HSDir relays and gradually pull to consensus.

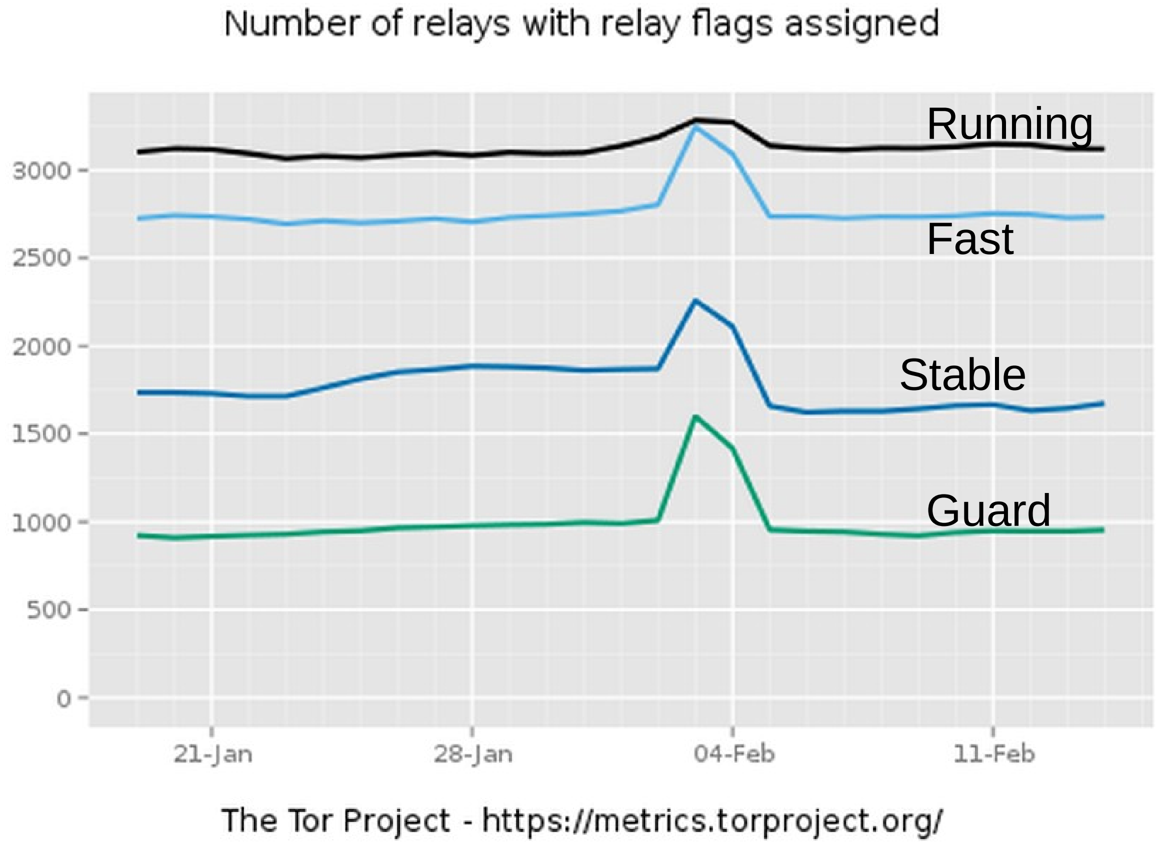


Harvest results

- We used 58 IP addresses from Amazon EC2 and spent 57 USD
- We collected 39824 unique onion addresses in 49 hours (on hidden wikis one can find ~2500 addresses only)
- Some interesting note: 12 onion addresses in the form silkroad*****.onion.

Side effect (flag assignment)

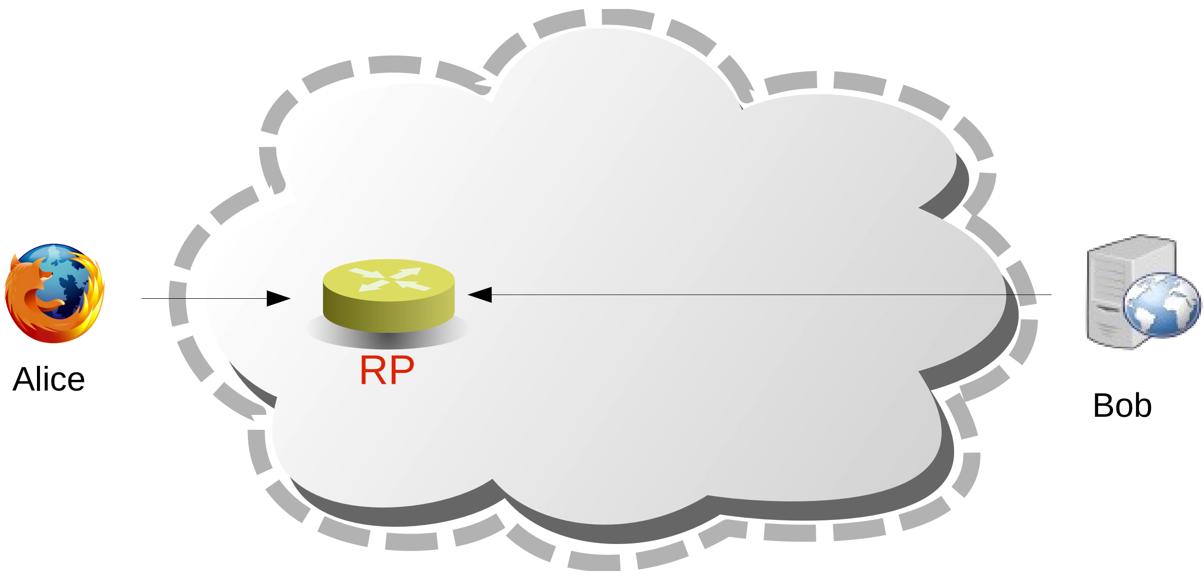
- Large number of shadow relays with bw <= 1 accelerated flag assignment.



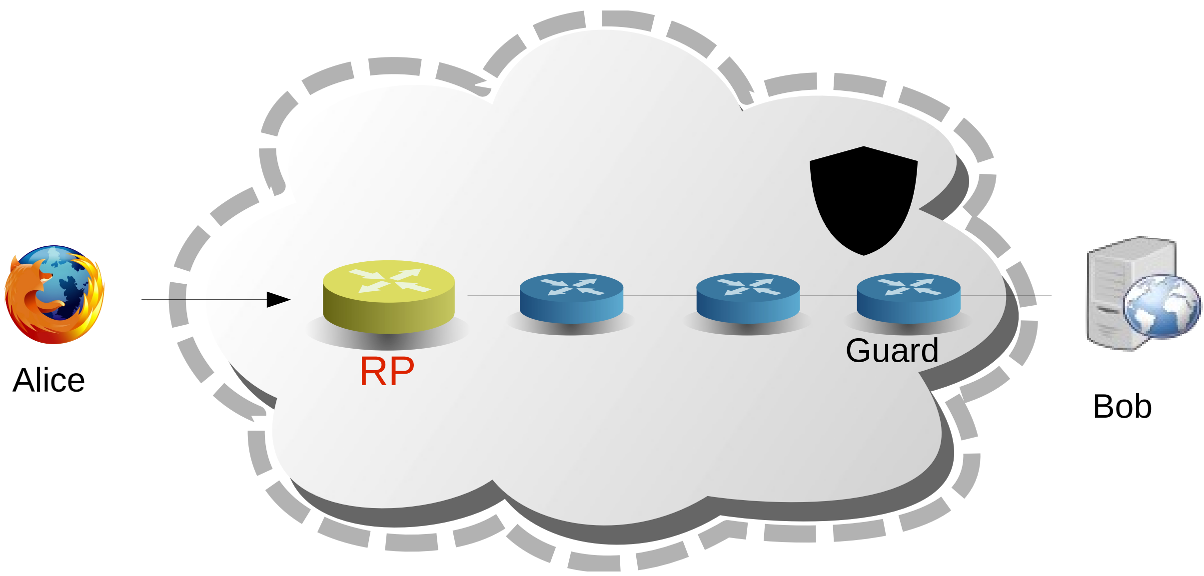
Outline

Tracking	✓
Denial of Service	✓
Collecting onion addresses	✓
Revealing Guard Nodes	
Deanonymisation	

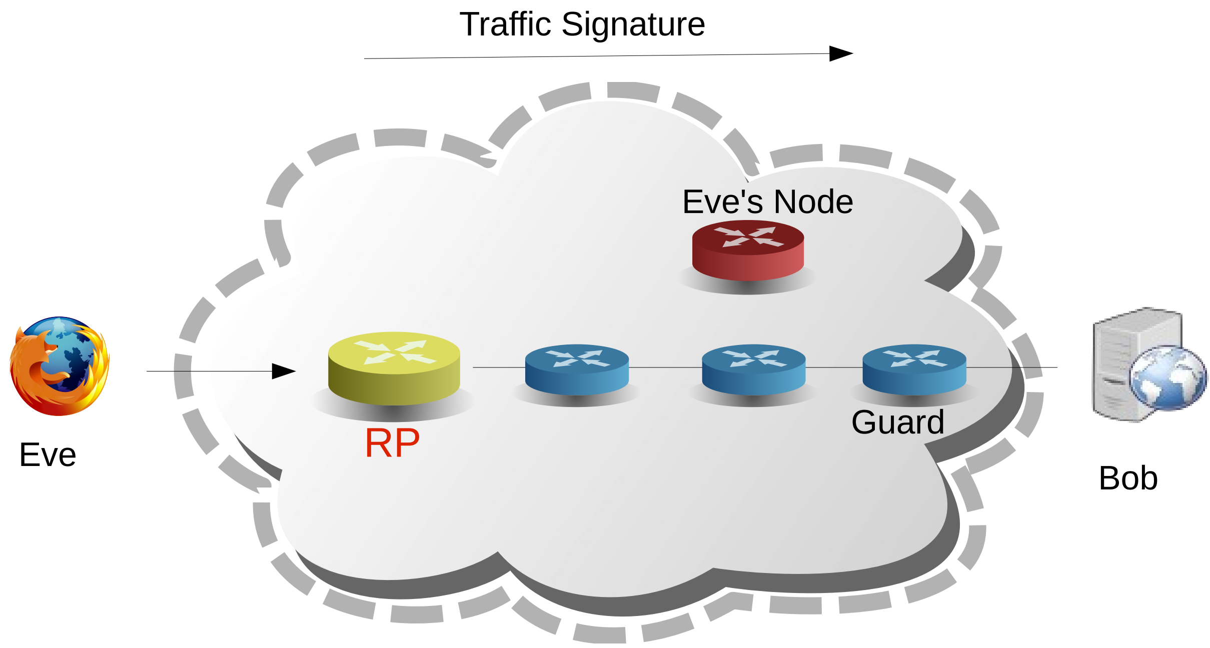
Revealing Guard Nodes



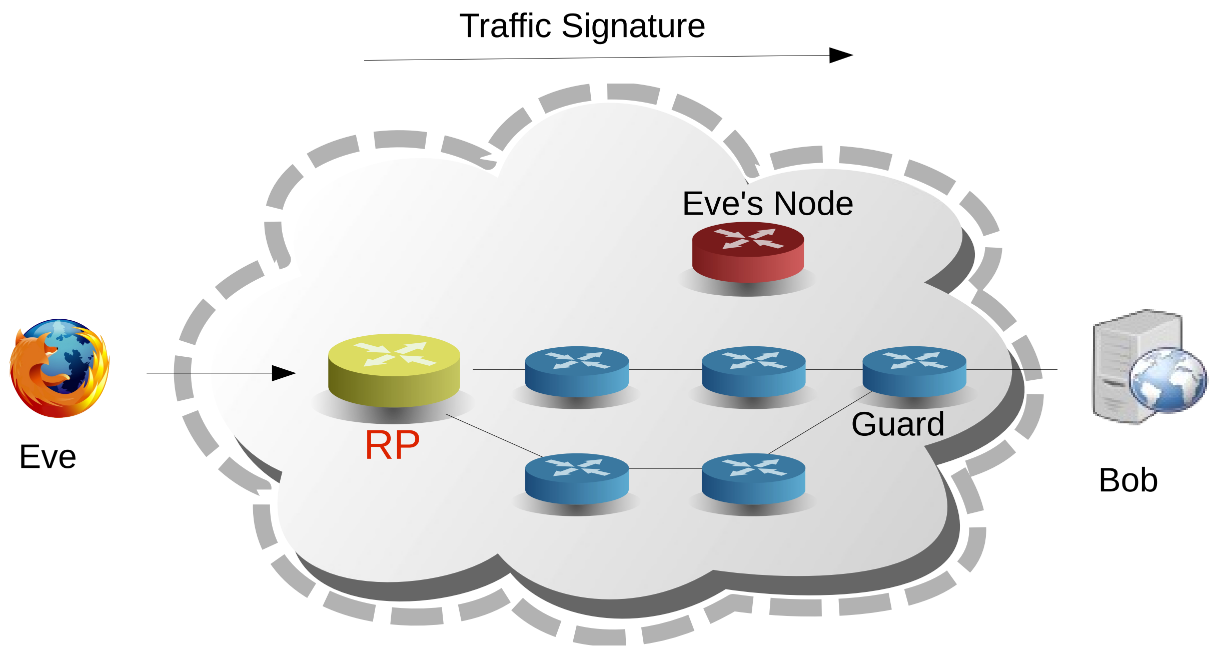
Revealing Guard Nodes



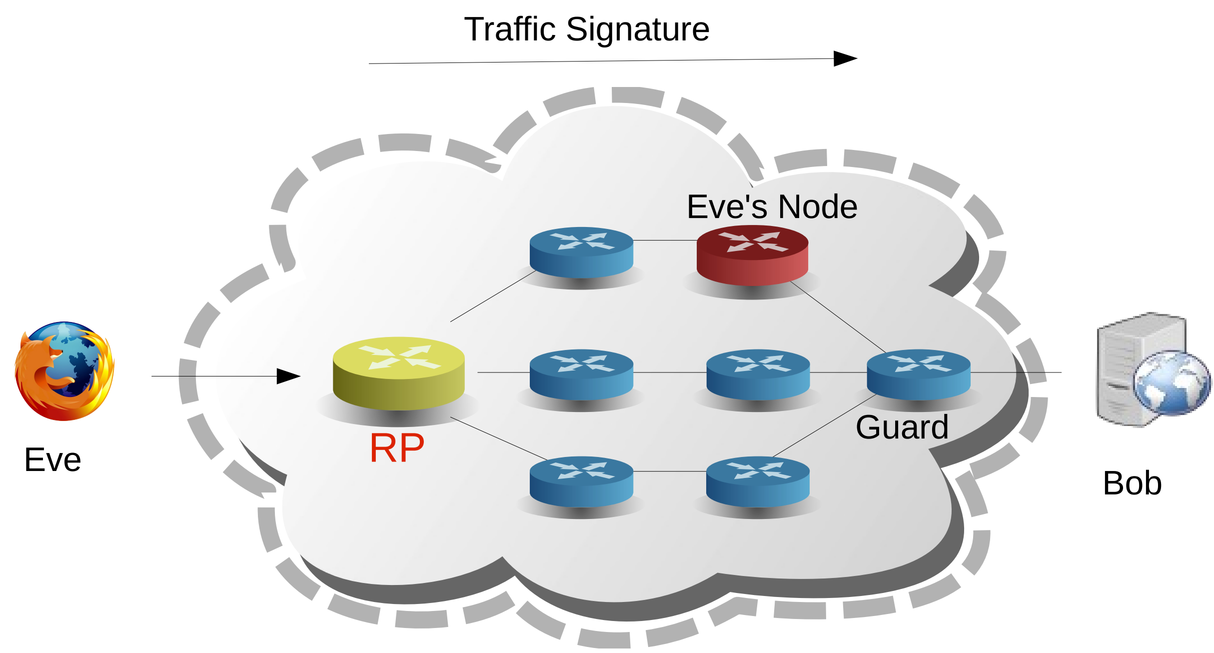
Revealing Guard Nodes



Revealing Guard Nodes

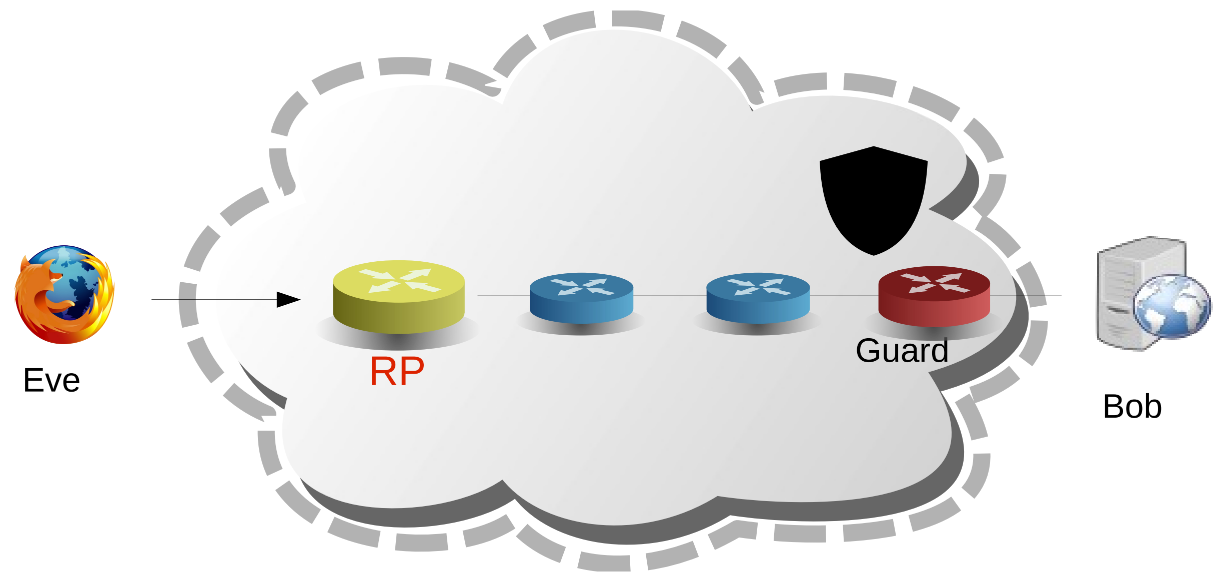


Revealing Guard Nodes

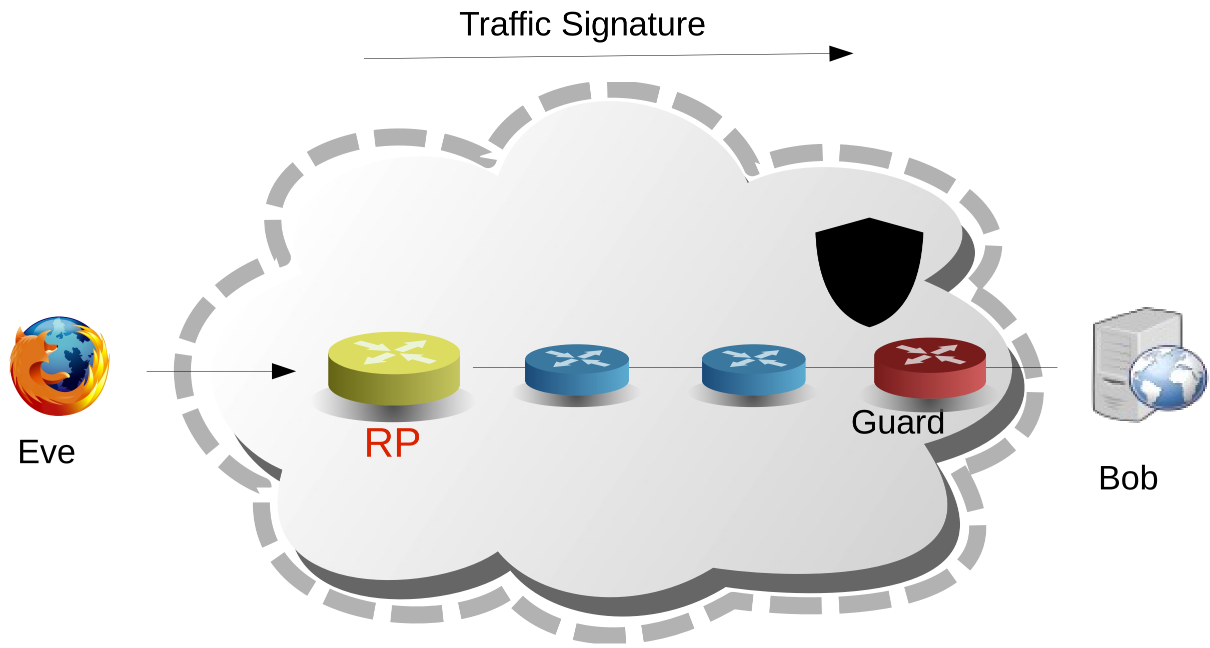


~40 minutes to reveal the guard nodes for a 5Mb/s node

Opportunistic deanonymisation



Opportunistic deanonymisation



How long does it take to become a Guard of a hidden service?

Opportunistic deanonymisation

- Rent a server for 60 USD per month => 0.6% probability to be chosen as a Guard.
- Deanonymisation **~150 hidden services per month** (for **60 USD** per month)
- By running 23 such servers, the probability to deanonymize **any** long-running **hidden service** within **8 months is 99%**. (~11 000 USD total).

Conclusions

Tracking	✓
Denial of Service	✓
Collecting onion addresses	✓
Revealing Guard Nodes	✓
Deanonymisation	• 150 addresses per month (60 USD) •Any HS (8 months+11000 USD) ✓

Support slide 1

- Triggered
  - #8243: Getting the HSDir flag should require more effort
  - #8243: Getting the HSDir flag should require more effort
- Related
  - Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor", WPES 2012
  - #8240: Raise our guard rotation period

    (patch to raise it to 9.5 month still pending)

Support slide 2

- Not included into the presentation
  - Finding guard nodes using topological properties
  - Bandwidth inflation

2.2 Session 2: The Best Rejects (how to get your paper published in a top conference)

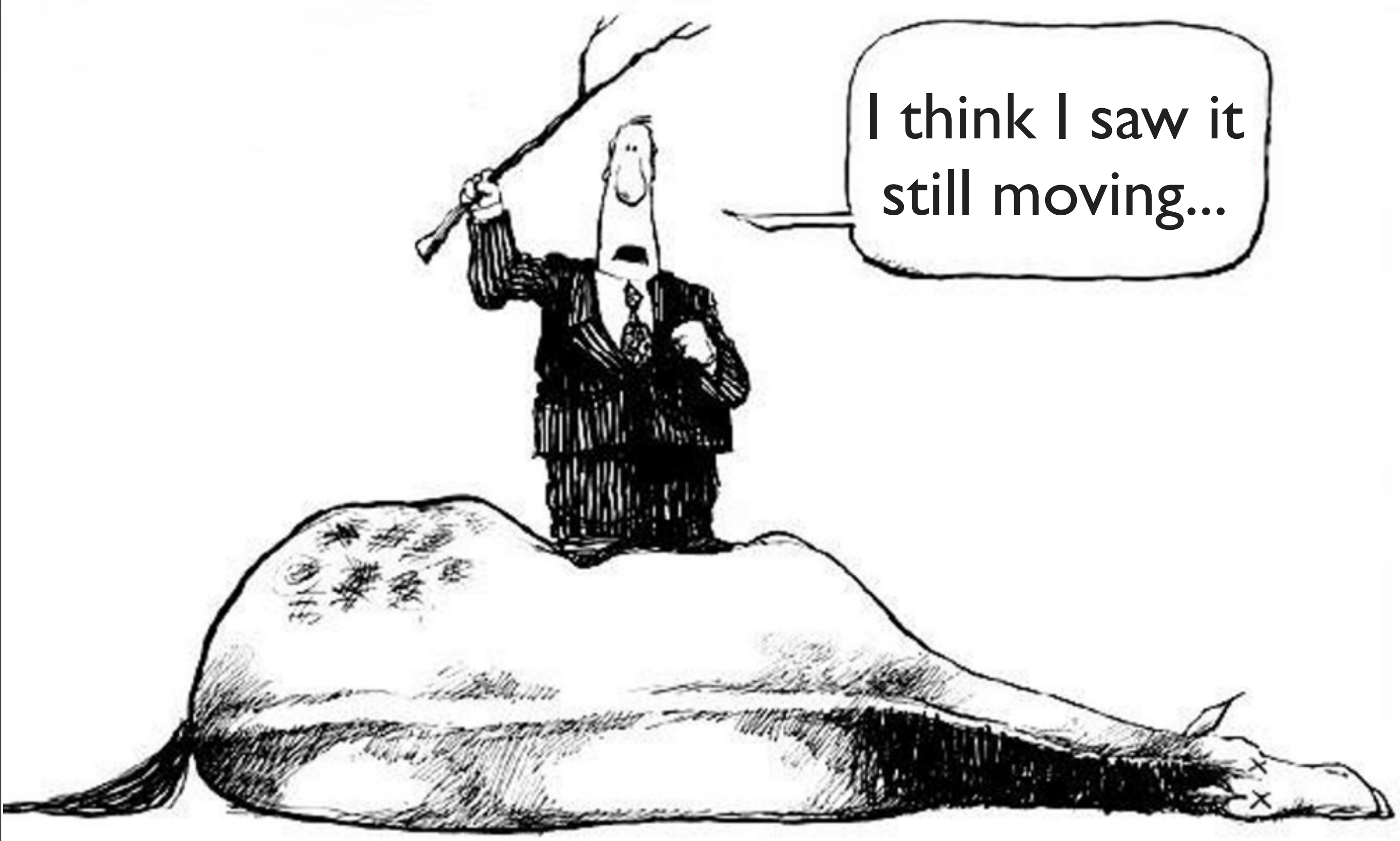
In this session, two very experienced EU researchers put themselves on the spot by addressing a topic rarely addressed, rejection of good research papers. They used as a case study one of their own papers that was rejected before being accepted in a top conference. In this way, students learned from experience how to get a paper published in a highly rated venue.

2.2.1 Lessons learned while publishing: Practical Timing Side Channel Attacks Against Kernel Space ASLR

Authors Ralf Hund, Carsten Willems, Thorsten Holz.

Speaker Thorsten Holz.

Paper Summary Due to the prevalence of control-flow hijacking attacks, a wide variety of defense methods to protect both user space and kernel space code have been developed in the past years. A few examples that have received widespread adoption include stack canaries, non-executable memory, and Address Space Layout Randomization (ASLR). When implemented correctly (i.e., a given system fully supports these protection methods and no information leak exists), the attack surface is significantly reduced and typical exploitation strategies are severely thwarted. All modern desktop and server operating systems support these techniques and ASLR has also been added to different mobile operating systems recently. In this paper, we study the limitations of kernel space ASLR against a local attacker with restricted privileges. We show that an adversary can implement a generic side channel attack against the memory management system to deduce information about the privileged address space layout. Our approach is based on the intrinsic property that the different caches are shared resources on computer systems. We introduce three implementations of our methodology and show that our attacks are feasible on four different x86-based CPUs (both 32- and 64-bit architectures) and also applicable to virtual machines. As a result, we can successfully circumvent kernel space ASLR on current operating systems. Furthermore, we also discuss mitigation strategies against our attacks, and propose and implement a defense solution with negligible performance overhead.



iseclab

hgi Horst Görtz Institut für IT-Sicherheit

2012 **Timeline** 2013

Ralf Hund, Carsten Willems, Thorsten Holz: "Practical Timing Side Channel Attacks Against Kernel Space ASLR"

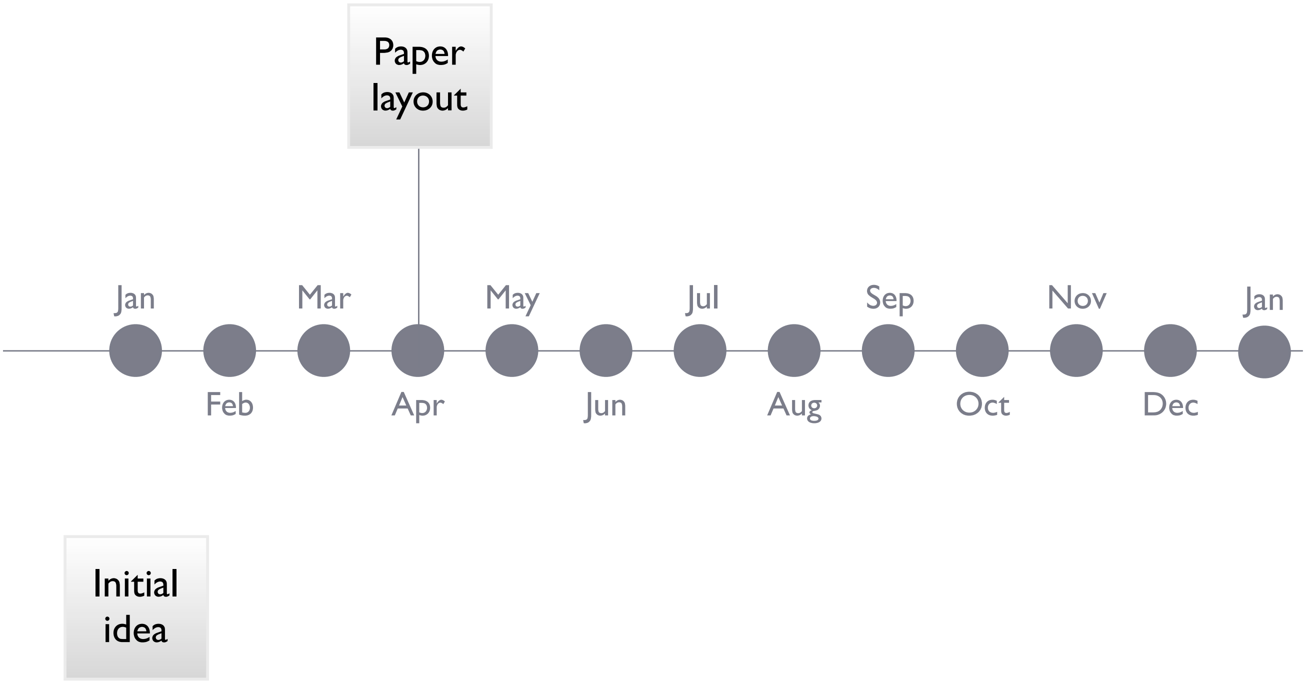
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan

Initial idea

2012 **Timeline** 2013

Timeline

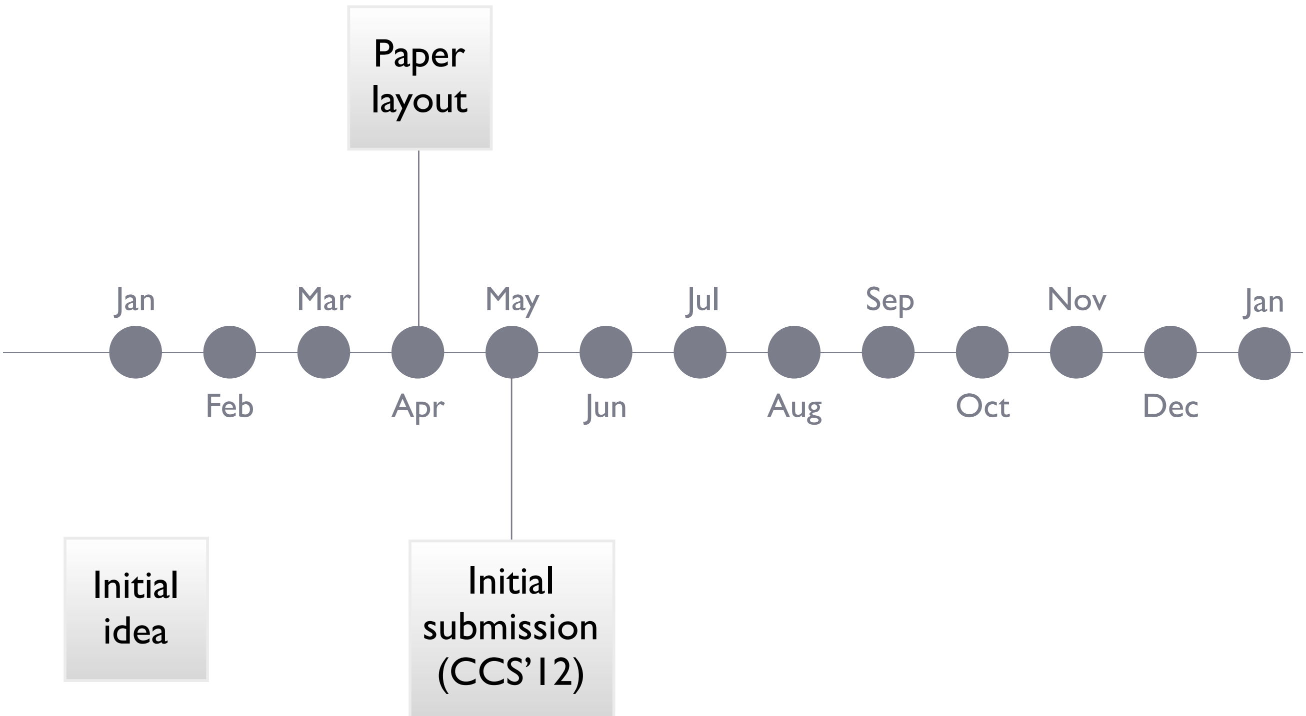
Ralf Hund, Carsten Willems, Thorsten Holz: “Practical Timing Side Channel Attacks Against Kernel Space ASLR”



2012 **Timeline** 2013

Timeline

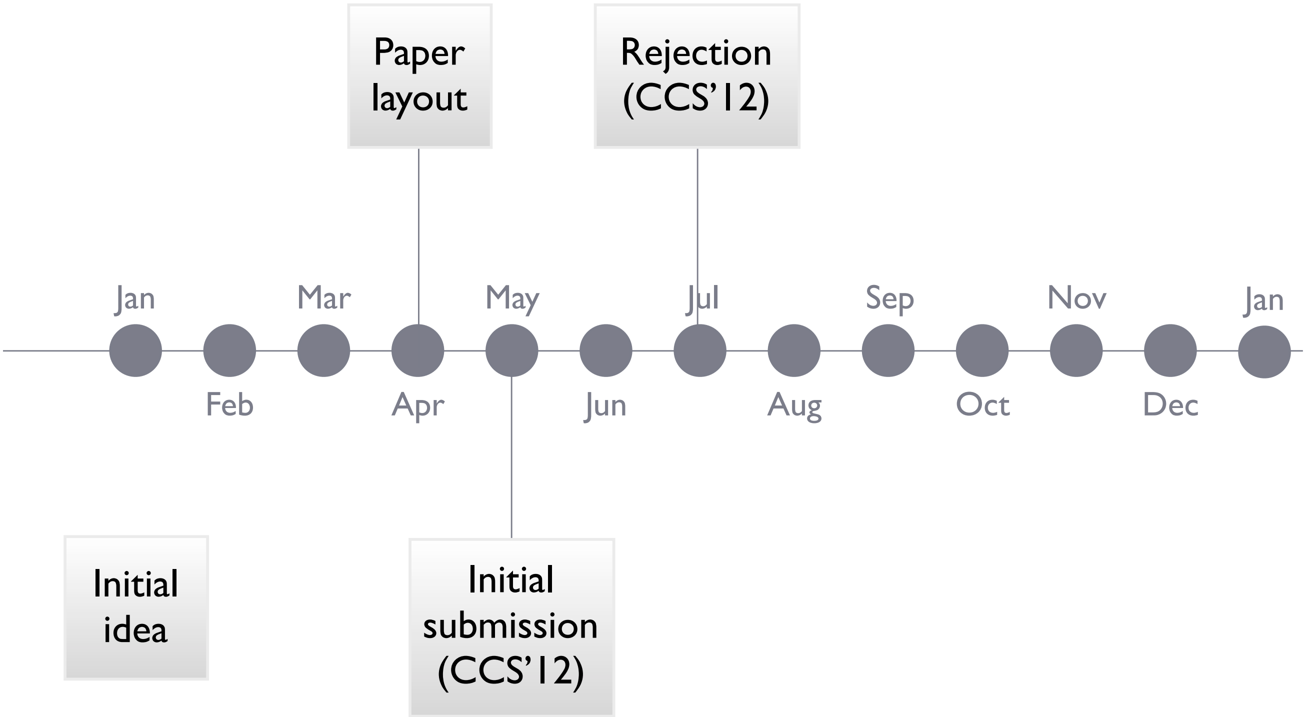
Ralf Hund, Carsten Willems, Thorsten Holz: “Practical Timing Side Channel Attacks Against Kernel Space ASLR”



2012 **Timeline** 2013

Timeline

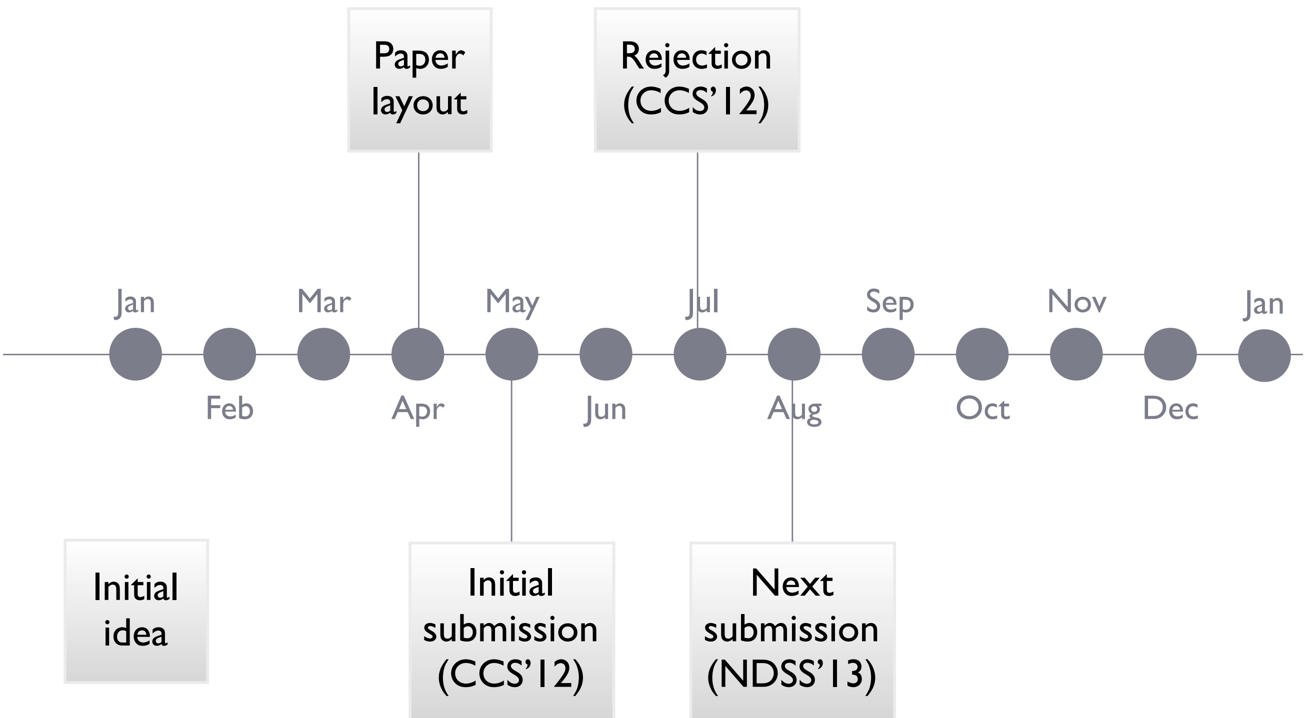
Ralf Hund, Carsten Willems, Thorsten Holz: “Practical Timing Side Channel Attacks Against Kernel Space ASLR”



2012 **Timeline** 2013

Timeline

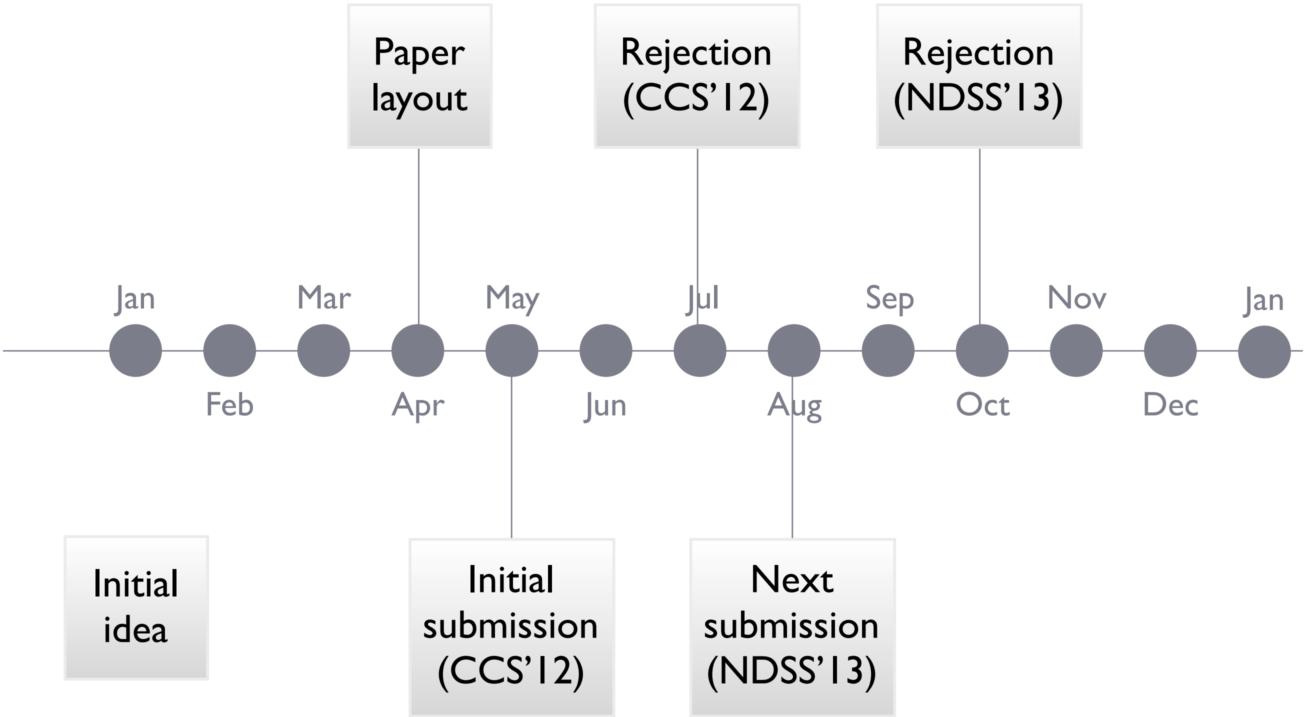
Ralf Hund, Carsten Willems, Thorsten Holz: “Practical Timing Side Channel Attacks Against Kernel Space ASLR”



Timeline

2012 2013

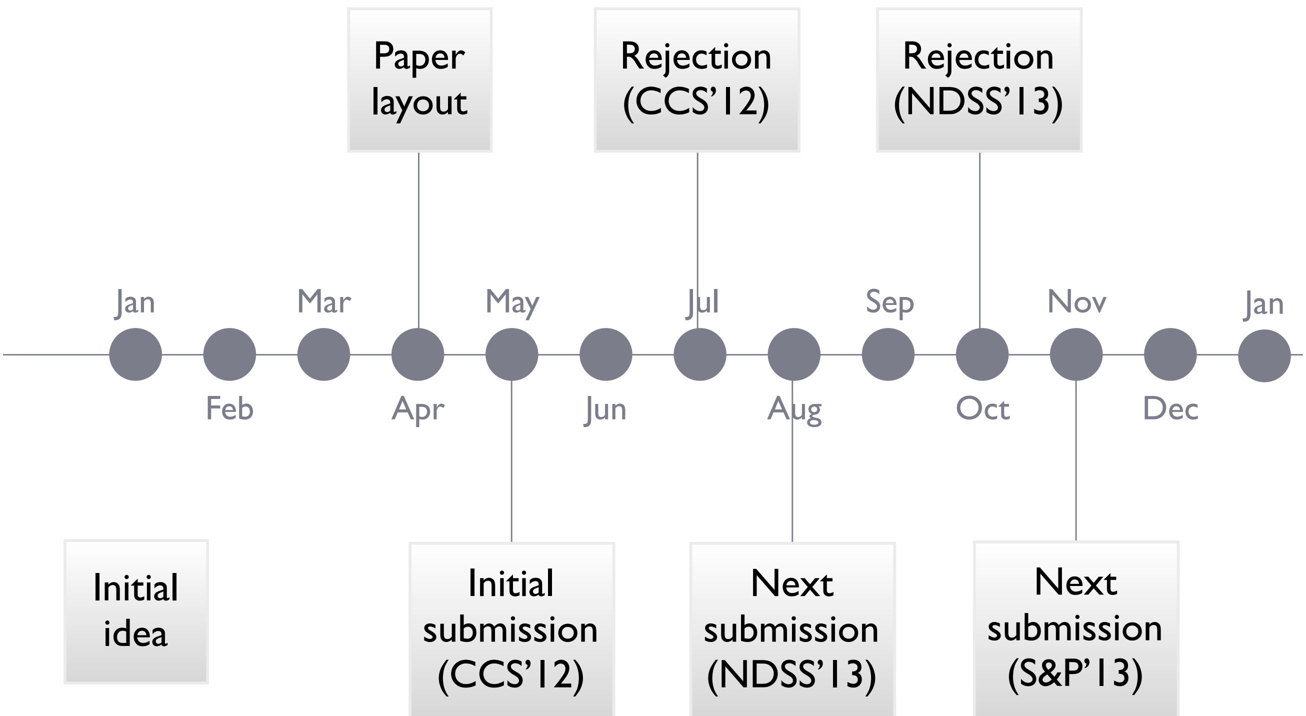
Ralf Hund, Carsten Willems, Thorsten Holz: "Practical Timing Side Channel Attacks Against Kernel Space ASLR"



Timeline

2012 2013

Ralf Hund, Carsten Willems, Thorsten Holz: "Practical Timing Side Channel Attacks Against Kernel Space ASLR"



Timeline

2012 2013

Ralf Hund, Carsten Willems, Thorsten Holz: "Practical Timing Side Channel Attacks Against Kernel Space ASLR"

Paper layout — Rejection (CCS'12) — Rejection (NDSS'13) — Acceptance (S&P'13)

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec, Jan

Initial idea — Initial submission (CCS'12) — Next submission (NDSS'13) — Next submission (S&P'13)

Timeline

2012 2013

Ralf Hund, Carsten Willems, Thorsten Holz: "Practical Timing Side Channel Attacks Against Kernel Space ASLR"

on 13) — Acceptance (S&P'13)

Jan, Feb, Mar, Nov, Dec, Jan

Initial idea — Next ubmission (S&P'13)

Finding Ideas

Systems Security
Ruhr-University Bochum

- Often a long and painful process!
- Discuss ideas with colleagues, even if the idea is still in a very early stage
  - Meet for a coffee and debate the topic
  - Regular brainstorming meetings
  - Take notes such that you can come back to topics
- Use this week to meet people working in your area!

Security Reading Group

Systems Security
Ruhr-University Bochum

- In my opinion, each research group should do this
- (Bi-)Weekly meeting where papers are discussed
  - Everyone reads the paper *in advance*
  - Somebody summarizes the paper
  - Discussion on strong and weak points
  - Potential follow-up?
  - Propose papers for next reading group

Security Reading Group

Systems Security
Ruhr-University Bochum

- In my opinion, each research group should do this
- (Bi-)Weekly meeting where papers are discussed
  - Everyone reads the paper *in advance*
  - Somebody summarizes the paper
  - Discussion on strong and weak points
  - Potential follow-up?
  - Propose papers for next reading group

Somebody needs to push this

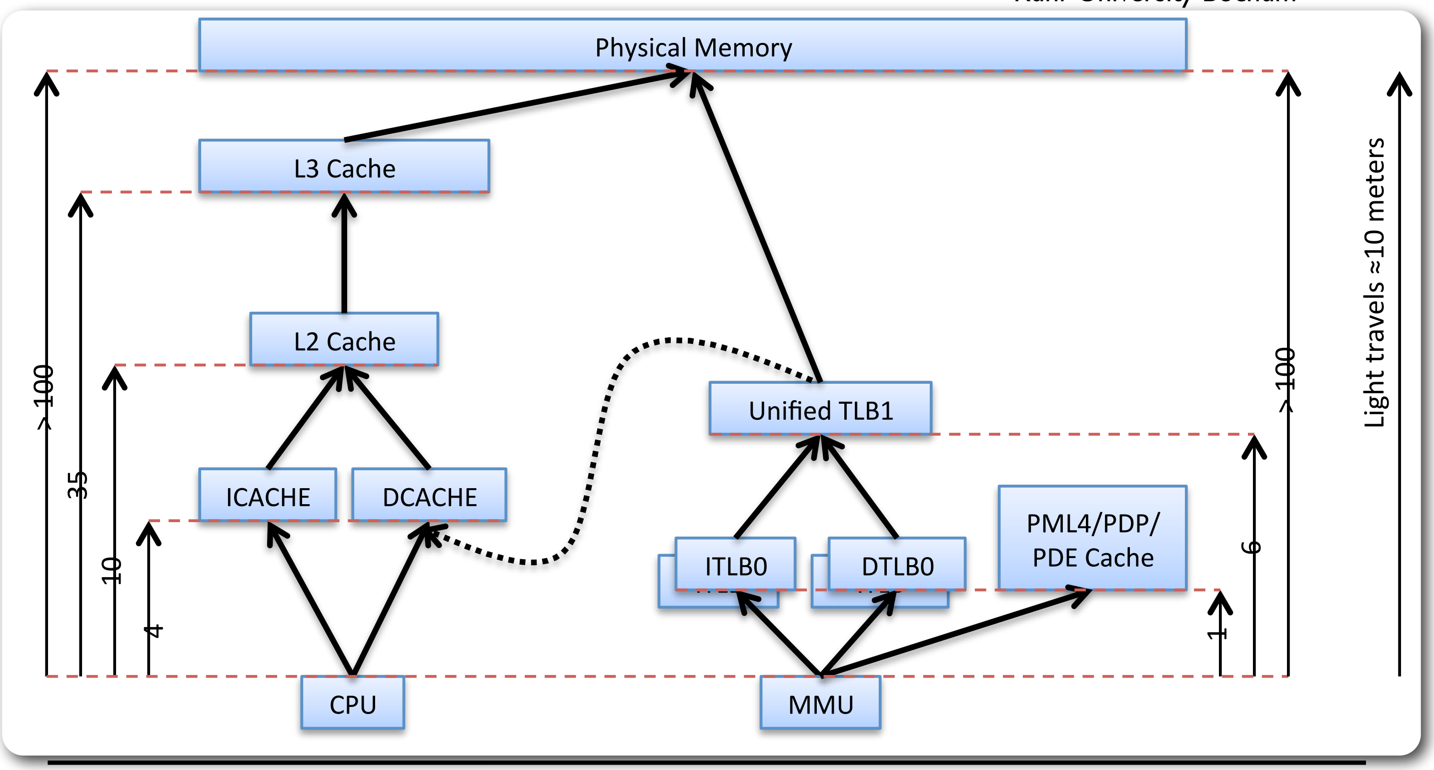
(Disclaimer: does not work for my group)

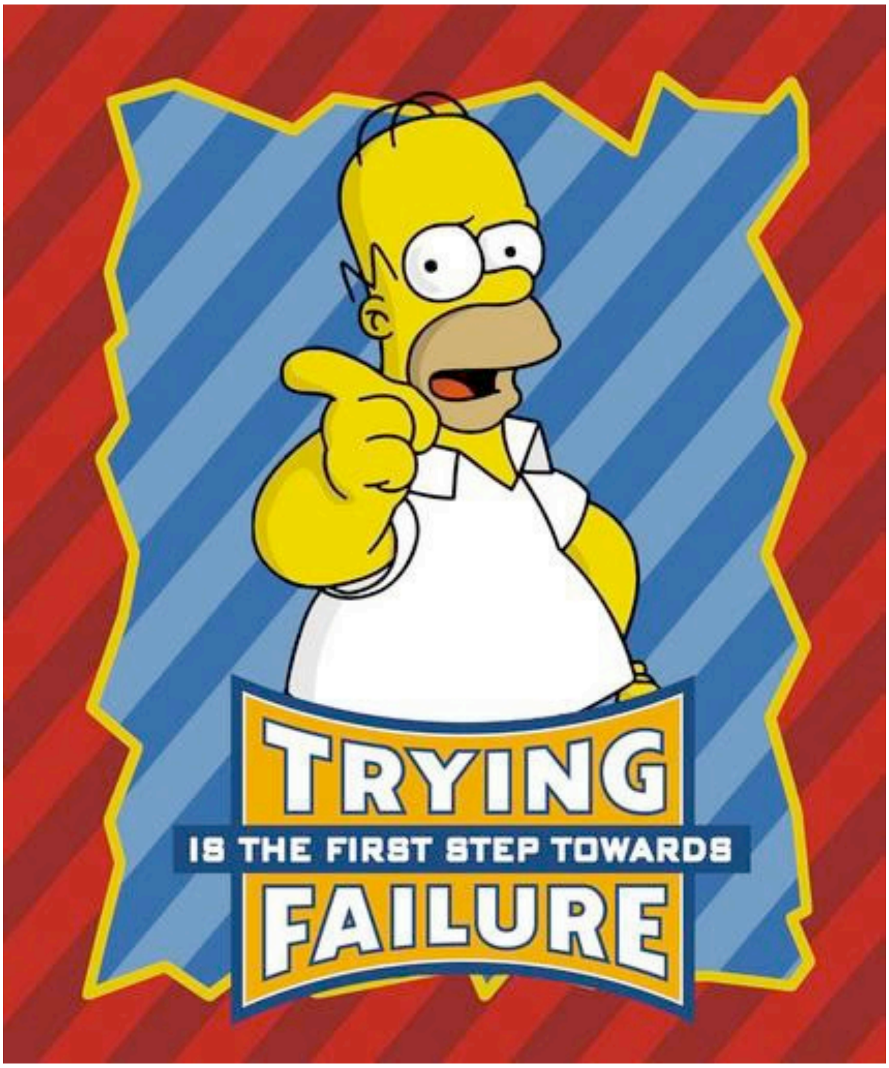
iSecLAB Side Channel Attacks Against Kernel Space ASLR • Lessons Learned Slide # 5
Mittwoch, 24. Juli 13

Our Case

- Took several weeks to come up with the topic
- At the beginning just a rough idea
  - How robust is kernel space ASLR on Windows?
  - Brute-force attacks are not feasible, what else can we do?
  - Are there timing difference when accessing specific memory locations?
  - ➢ Try to precisely measure time ⇒ *side channel attack*

Our Case



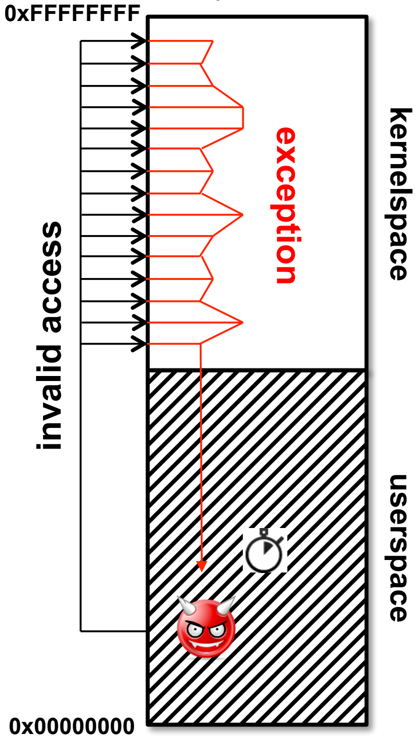


Implementation

- Often a long and painful process!
- Start with small examples to test general feasibility
  - Scalability, performance, memory consumption, ... can be improved later on
  - Yet the example should be more than a toy
  - Manual confirmation/testing often needed, automation then comes into play
- Maybe get help, work in teams

Approach #1

- Abstract idea
  - Access kernel space addresses two times
  - Measure time duration until exception delivered
- One probe of entire kernel space takes ≈2 seconds (32-bit)
- 2^{19} (≈ 500 000) measurements



Our Case

- Initial tests promising, but many obstacles appeared
  - Implementation was challenging
  - Lots and lots of system details needed, developer manuals were (typically) only reliable source
  - Very low-level analysis (e.g., RE of undocumented hash function used in Intel Sandybridge CPUs to distribute the cache among different cores)
  - ➢ Kudos to Ralf and Carsten!

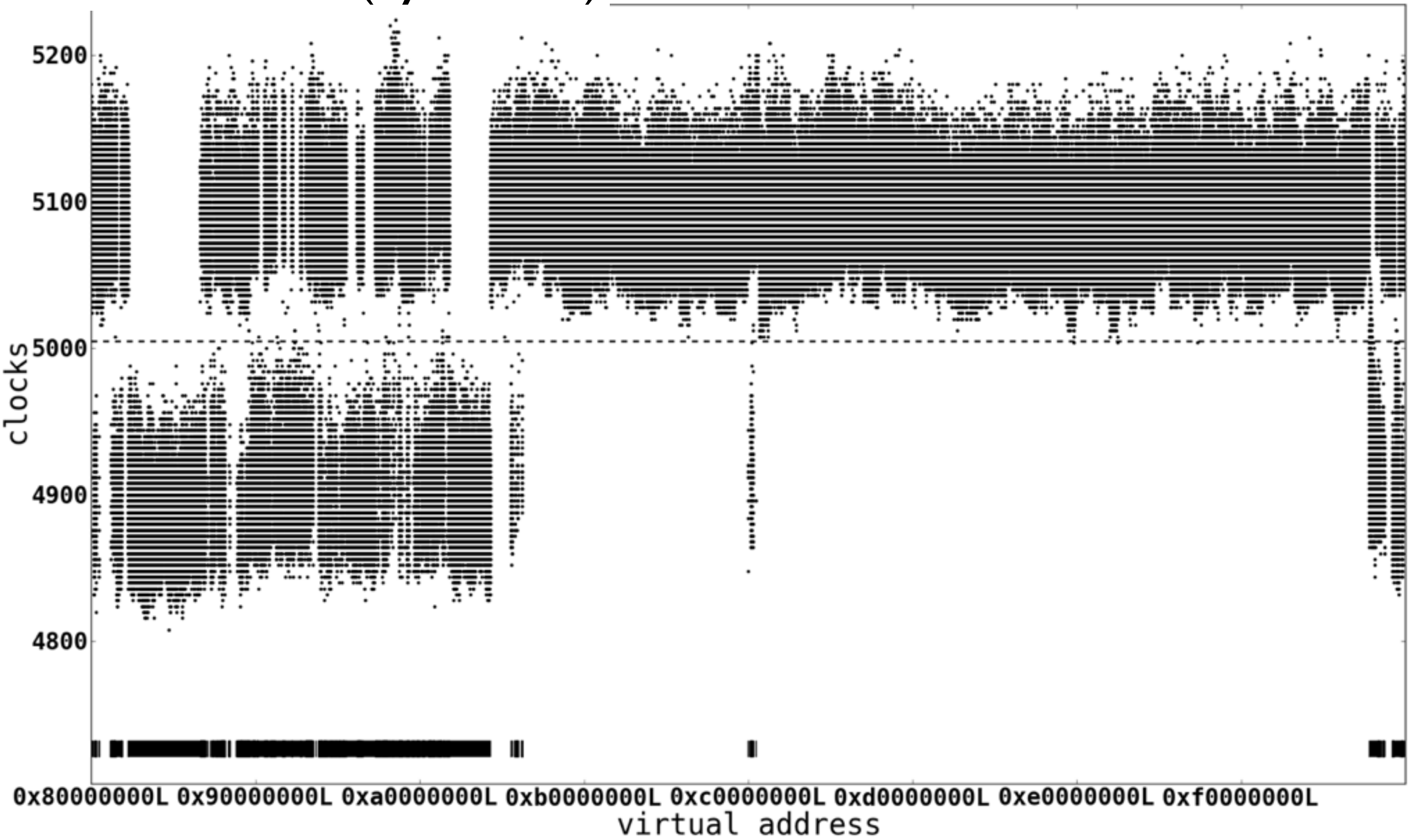
Often unclear if project was doable at all, persistence needed!

Evaluation

- Important aspect of (systems) papers
  - Demonstrate that your work is valuable
  - Compare your work against existing systems (if available) and demonstrate improvements
  - Often hard to properly compare systems (e.g., which analysis report is "better"?)
- Soundness and false negatives are hard to measure

Our Case

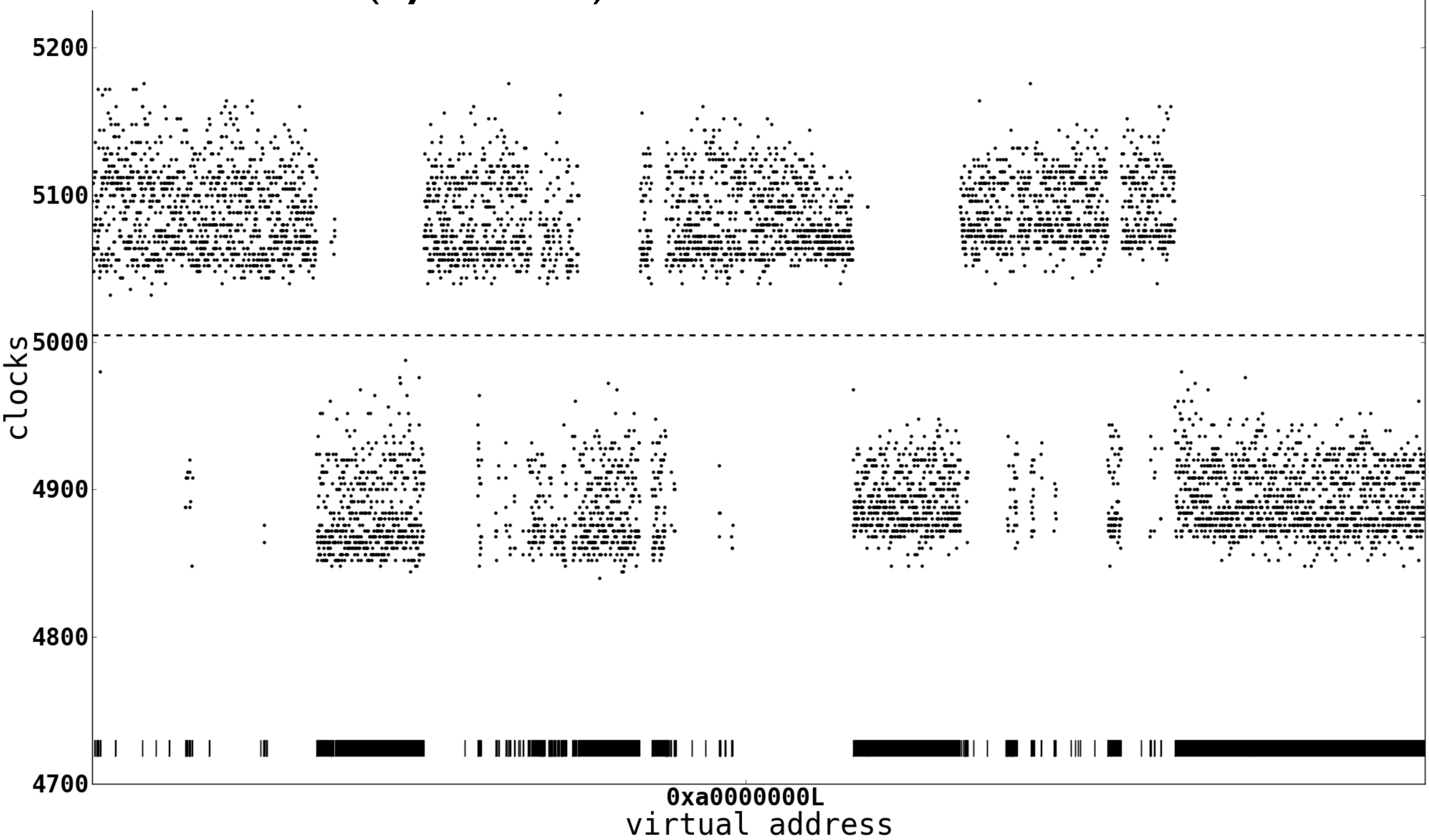
Intel i7-950 (Lynnfield)



Our Case

Intel i7-950 (Lynnfield) - Zoomed in

Systems Security



Writing

- Structure of papers is often similar
  - Generic structure: introduction, background, overview, implementation details, evaluation, related work, conclusion, (appendix), references
  - Related work early on?
- Get feedback from your advisor, you will learn how to write over time
- Polish papers as good as possible (as Nick already said)
- Reading good papers helps ⇒ security reading group

We regret to inform you... [CCS'12]

Review 1:
It is a real problem in real systems. [...] It would be more convincing if the exploits were carried out in a more realistic setting. [...] I recommend accept because the finding needs to be shared with the community

We regret to inform you... [CCS'12]

Review 1:
It is a real problem in real systems. [...] It would
be
ou
ac
w

Review 2:
The paper provided a great amount of technical details [...] the threat model is not consistent [...] *generic* seems farfetched [...] a more thorough literature review on previous studies [...] a few minor complaints on the basic assumptions in the paper

We regret to inform you... [CCS'12]

Review 1:
It is a real problem in real systems. [...] It would
be
ou
ac
w

Review 2:
The paper provided a great amount of technical
de
[...
th
[...
as

Review 3:
Do we really need more evidence that ASLR is an ineffective defense? To a certain extent this is beating a dead horse [...] cleverness is all in the idea of using timing channels [...] details of the attack are actually not very well explained

Revision #1

- Improved implementation
  - Linux
  - 64 bit CPUs
- Performed more experiments
- Revised complete paper
  - Took reviewers' comments into account
  - Technical description revised and extended
- ➢ Significantly better paper!

We regret to inform you... [NDSS'13]

Review 1 (accept):
do not talk of noise that might be introduced by concurrently running processes on the system [...] The evaluation could have been better [...] paper is well written, results look very good

We regret to inform you... [NDSS'13]

Review 1 (accept):

dc
by
sy
be
ve

Review 2 (borderline):

The idea is original, implementation is laudable, although there are still some weak points as identified above. The paper is well-written, but I suggest the authors compact the background section and add some discussion about their limitations regarding the weaknesses.

We regret to inform you... [NDSS'13]

Systems Security
Ruhr-University Bochum

Review 1 (accept):

dc
by T
sy al
be id
ve su

Review 2 (borderline):

Review 3 (weak reject):

Weaknesses: Attack Scenario, Missing Real-World Example Exploit, Time, Noise, Related Work, Cache Probing, ...

section and add some discussion about their
limitations regarding the weaknesses.

Side Channel Attacks Against Kernel Space ASLR • Lessons Learned Slide # 18
Mittwoch, 24. Juli 13

We regret to inform you... [NDSS'13]

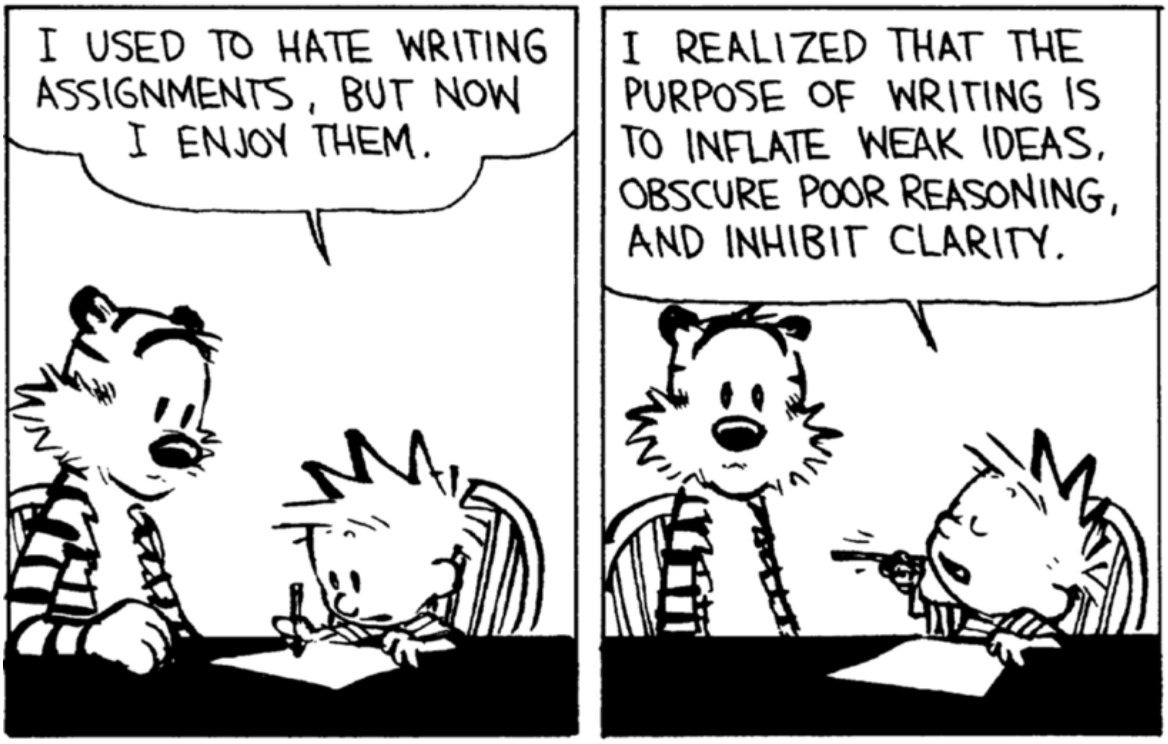
Review 1 (accept):

Review 2 (borderline):

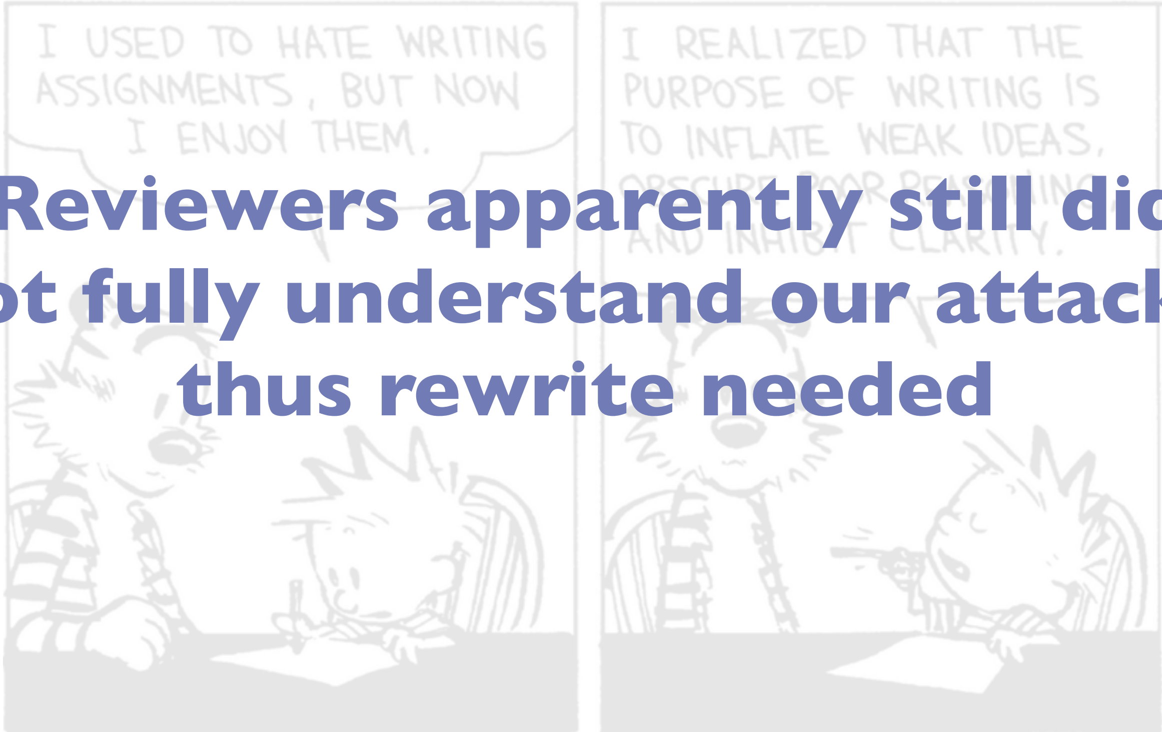
Review 3 (weak reject):

Review 4 (weak reject):
treatment of details in the paper is also unbalanced [...] In conclusion, although the paper is quite interesting, improvements need to be made for it to be accepted. [...] the selection of related work is quite limited

Revision #2



Revision #2



Reviewers apparently still did not fully understand our attacks, thus rewrite needed

... is delighted to inform you [S&P'13]

Review 1 (borderline):

Review 2 (accept):

Review 3 (accept):

Review 4 (weak accept):

Review 5 (weak accept):

Breaking ASLR in a matter of seconds to minutes is very valuable. Yes, if the OS randomizes more this would take longer but I agree with the authors that the proposed side channel is a high quality channel and can more or less give the answer even for 64-bit full randomization.

Lessons Learned

Systems Security
Ruhr-University Bochum

- Finding ideas, implementing them and finally evaluating everything can be a cumbersome process
- You will improve with your writing over time
- Take reviews seriously and revise paper accordingly
  - Do not stop working on a project after submission (no "fire and forget", although we also often do this)
  - Treat it as an ongoing project, paper submissions are only snapshots/milestone for the long term

Questions?

Systems Security
Ruhr-University Bochum

Contact:
Prof. Thorsten Holz
thorsten.holz@rub.de

More information:
http://syssec.rub.de
https://moodle.rub.de

2.2.2 Lessons learned while publishing: Dowsing for overflows: A Guided Fuzzer to Find Buffer Boundary Violation

Authors Istvan Haller, Asia Slowinska, Matthias Neugschwandtner, Herbert Bos.

Speaker Herbert Bos.

Paper Summary Dowser is a "guided" fuzzer that combines taint tracking, program analysis and symbolic execution to find buffer overflow and underflow vulnerabilities buried deep in a program's logic. The key idea is that analysis of a program lets us pinpoint the right areas in the program code to probe and the appropriate inputs to do so.

Intuitively, for typical buffer overflows, we need consider only the code that accesses an array in a loop, rather than all possible instructions in the program. After finding all such candidate sets of instructions, we rank them according to an estimation of how likely they are to contain interesting vulnerabilities. We then subject the most promising sets to further testing. Specifically, we first use taint analysis to determine which input bytes influence the array index and then execute the program symbolically, making only this set of inputs symbolic. By constantly steering the symbolic execution along branch outcomes most likely to lead to overflows, we were able to detect deep bugs in real programs (like the nginx webserver, the inspircd IRC server, and the ffmpeg videoplayer). Two of the bugs we found were previously undocumented buffer overflows in ffmpeg and the poppler PDF rendering library.

How to get your paper on

Dowsing for Overflows

A Guided Fuzzer to Find Buffer Boundary Violations

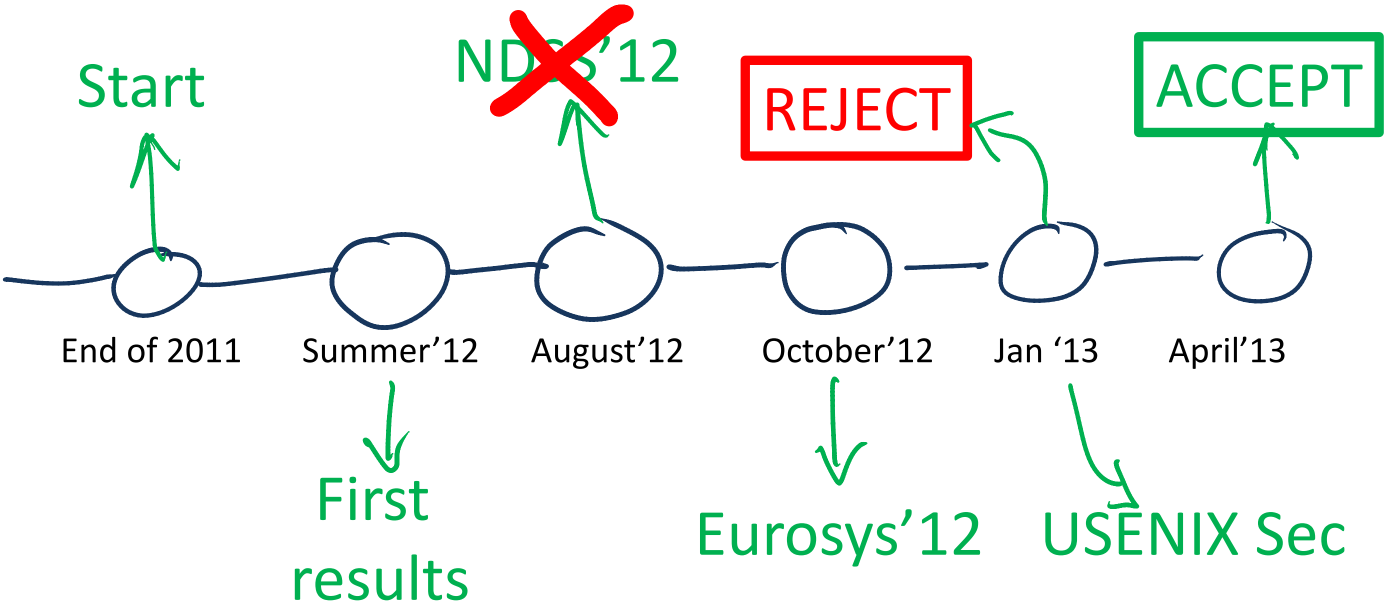
accepted

Istvan Haller
Asia Slowinska
Matthias Neugschwandtner
Herbert Bos

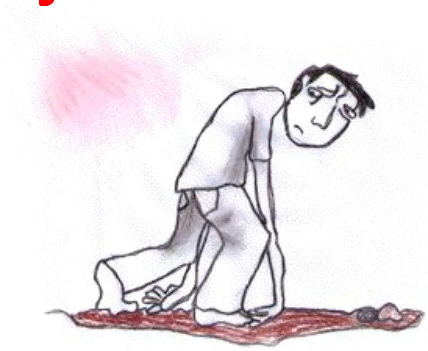
Herbert Bos
VU University Amsterdam

a great reject

Timeline



Everyone gets papers rejected



Typically something like

- Strengths:
  - represents a nice engineering effort
  - the system comes with a working prototype.
- Weaknesses:
  - it is not clear that this represents a significant advancement of the state of art in this area of research over and beyond the first generation papers on X, Y, and Z

Everyone gets papers rejected

Typically something like

- Strengths:
  - interesting set of heuristics for targeting buffer overflows
- Weaknesses:
  - the techniques are not clearly presented and justified
  - weak experimental evaluation, which provides little insight into the benefits of the different heuristics employed

Everyone gets papers rejected

Occasionally:

- Weaknesses: this system attempts to achieve something extremely undesirable.
- Strengths: It fails to achieve its undesirable goal."

Everyone gets papers rejected

E.W. DIJKSTRA

"Goto Statement Considered Harmful." This paper tries to convince us that the well-known goto statement should be eliminated from our programming languages or, at least (since I don't think that it will ever be eliminated), that programmers should not use it. It is not clear what should replace it. The paper doesn't explain to us what would be the use of the "if" statement without a "goto" to redirect the flow of execution: Should all our postconditions consist of a single statement, or should we only use the arithmetic "if," which doesn't contain the offensive "goto"?
And how will one deal with the case in which, having reached the end of an alternative, the program needs to continue the execution somewhere else?

The author is a proponent of the so-called "structured programming" style, in which, if I get it right, gotos are replaced by indentation. Structured programming is a nice academic exercise, which works well for small examples, but I doubt that any real-world program will ever be written in such a style. More than 10 years of industrial experience with Fortran have proved conclusively to everybody concerned that, in the real world, the goto is useful and necessary: its presence might cause some inconveniences in debugging, but it is a de facto standard and we must live with it. It will take more than the academic elucubrations of a purist to remove it from our languages.
Publishing this would waste valuable paper: Should it be published, I am as sure it will go uncited and unnoticed as I am confident that, 30 years from now, the goto will still be alive and well and used as widely as it is today.

Confidential comments to the editor: The author should withdraw the paper and submit it someplace where it will not be peer reviewed. A letter to the editor would be a perfect choice: Nobody will notice it there!

Often your work is excellent

- But you are selling it badly
- Writing a good motivation is very hard
  - Ask for help. Learn.
  - Take your reading group seriously

- Some things really simple but you don't do them
  - Topic sentences
  - Readable figures
  - Experiments that validate the claims
  - Treat related work fairly
  - Mention weaknesses

So, what's up with Dowser?

Dowsing is a type of divination used to find ground water buried treasure, rare gemstones, and now also bugs...

Where's the fire?

- Buffer overflows are still a top 3 threat!
  - Triggered under rare conditions
- Applications grow rapidly
  - Automated testing doesn't scale!

Security testing today

Surely, bugs can be anywhere!

- Can they?
- What do we need for a buffer overflow?
  - Buffer
  - Accesses to that buffer
  - Loop
- We can look for these properties *a priori*!

Moreover…

- All loops are created equal, but some loops are more equal than others
  - Complex code is buggier than simple code
  - …

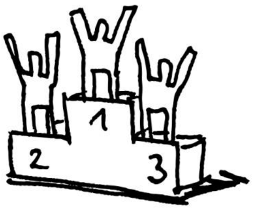
Buffer underrun in nginx

```
while (p <= r->uri_end)
    switch (state)
    case sw_usual: *u++ = ch; ...
    case sw_slash: *u++ = ch; ...
    ...
    case sw_dot: *u++ = ch; ...
        if (ch == '/') u--; ...
    case sw_dot_dot: *u++ = ch; ...
        if (ch == '/') u -= 4; ...
    ...
```

400 lines of code that make your head hurt

Idea: dowse for vulnerabilities

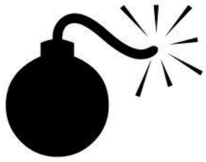
- Don’t try to verify all inputs
  - Focus the search for bugs on small and “potentially suspicious” code fragments



1. Identify places in the code that *might* look fishy

2. Perform a detailed analysis of these candidates
“Symbolic execution”

“Spot checking”

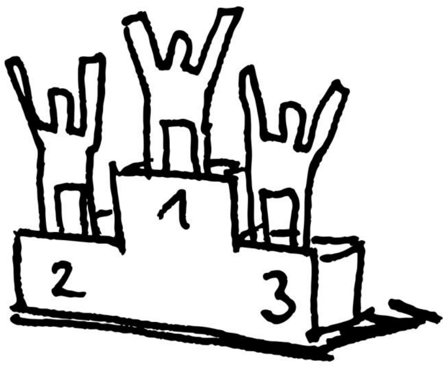


3. When applicable, find an input exploiting the vulnerability

1. Identify places are likely to have bugs

Buffer overflows in software

- Requirements:
  - An array
  - A pointer accessing the array
  - In a loop
- Our strategy:
  - Rank based on complexity: evaluate the complexity of array pointer operations, e.g.,
    - p++?
    - p+=4, p+=1, and p−=4?



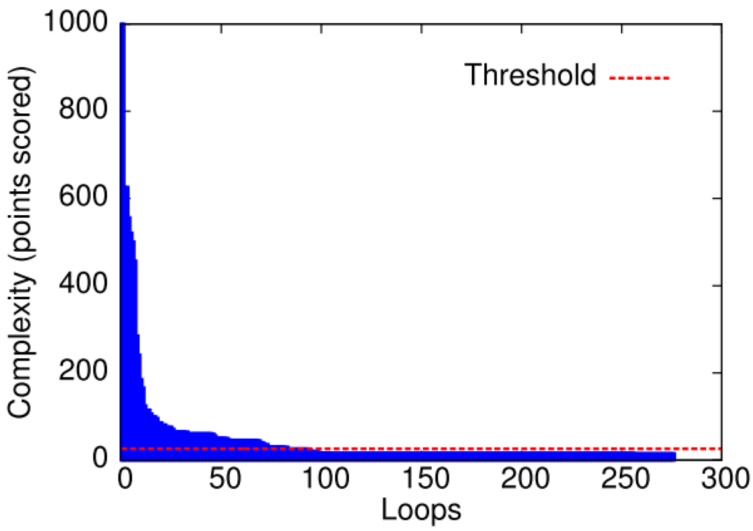
How do we rank?

- We score based on
  - Instructions
  - Different constants
  - Pointer casts
  - ….

Instructions	Points
Array index manipulations	
Basic index arithmetic instructions, i.e., addition, and subtraction	5
Other index arithmetic instructions, e.g., division, multiplication, shift, and xor	10
Different constant values	10
Constants involved in accessing fields of structures	0
Numerical values determined outside the loop	30
Non-inlined functions returning non-pointer values	500
Data movement instructions	0
Pointer manipulations	
Loading a pointer calculated outside the loop, e.g., an operation retrieving the base pointer of an object	0
`GetElemPtr` – an LLVM instruction that computes a new pointer from a base and offset(s)	5
Pointer cast operations, i.e., `PtrToInt` and `IntToPtr`	100

Does that work?!

- Consider nginx…
  - 70% of loops have minimal complexity
  - Example loop is in the top 5%



2. Symbolic execution

- Aim: find input that exercises the target
- Intuition:
  - model the behavior of a program using symbols instead of concrete values
  - Find an input that satisfies the model

2. Symbolic execution

- Example: let’s model the speed of a car

	Concrete values	Symbolic values
	115 km/h	$100 \leq v \leq 120$ km/h
	115 km/h	$0 <= v <= 120$ km/h
	250km/h	$v >= 0$ km/h

2. Symbolic execution

- Example: let's model the speed of a car

For code we do exactly the same:
- mark all input as symbolic, e.g., from the user/network
- execute the program using the symbols
- collect constraints
- solve the constraints to see if they can be satisfied

Concrete values	Symbolic values
115 km/h	$100 \le v \le 120$ km/h
115 km/h	0 <= v <= 120 km/h
155 km/h	v >= 0 km/h

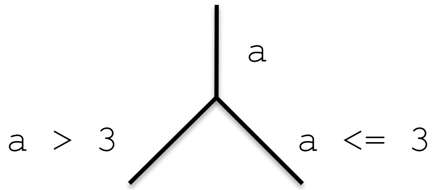
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass?(0);
}
```

a

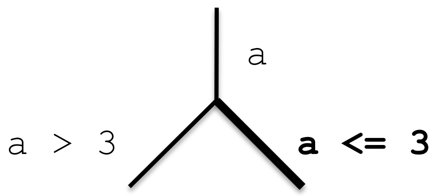
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



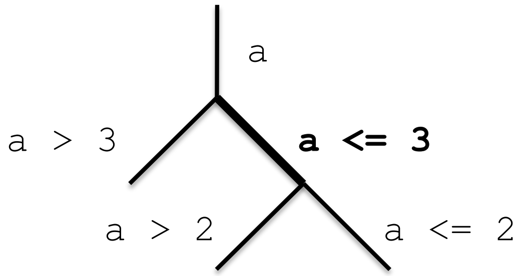
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



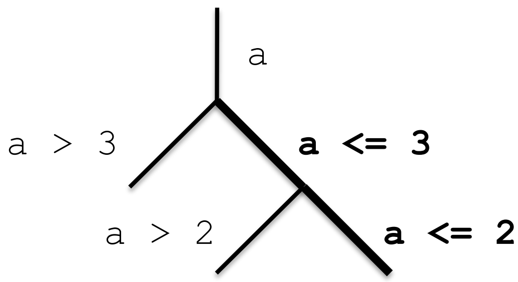
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



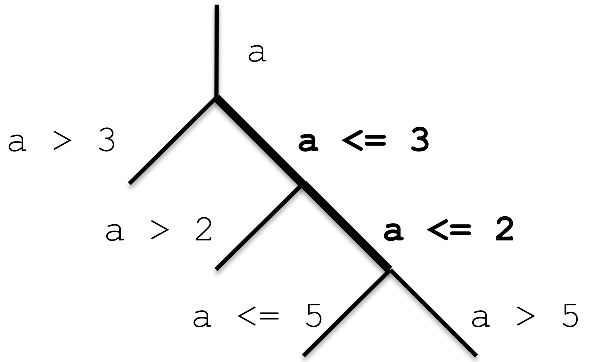
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



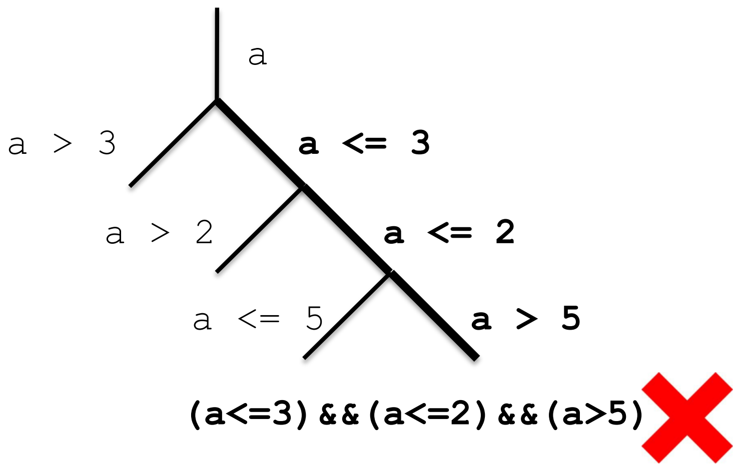
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



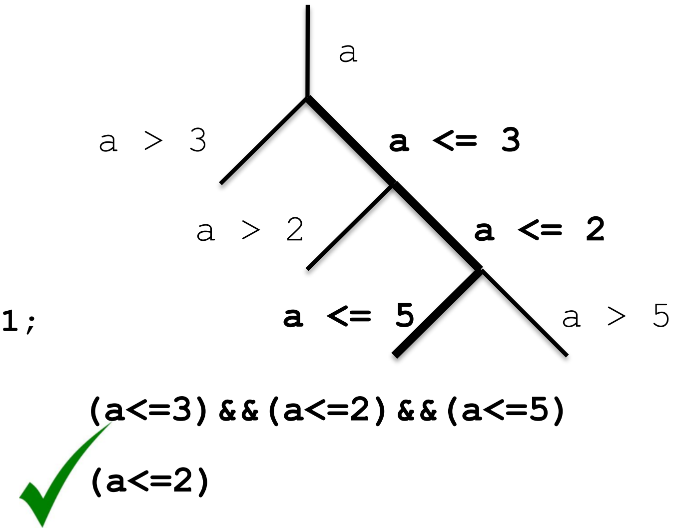
2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                ass? (0);
}
```



2. Symbolic execution

```
if (a > 3)
        exit(0);

if (a > 2) {
        do_something0;
} else {
        if (a <= 5)
                do_something1;
        else
                assert(0);
}
```



2. Symbolic execution

- Does not scale!
  - The number of states grows exponentially, so the analysis of a complex program can take ages!
  - E.g., nginx vulnerability not found within 8 hours

- Use taint analysis to find out what inputs should be symbolic

Nginx

Long input with multiple tokens.

GET /long/path/file HTTP/1.1
Host: thisisthehost.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 1337

Nginx

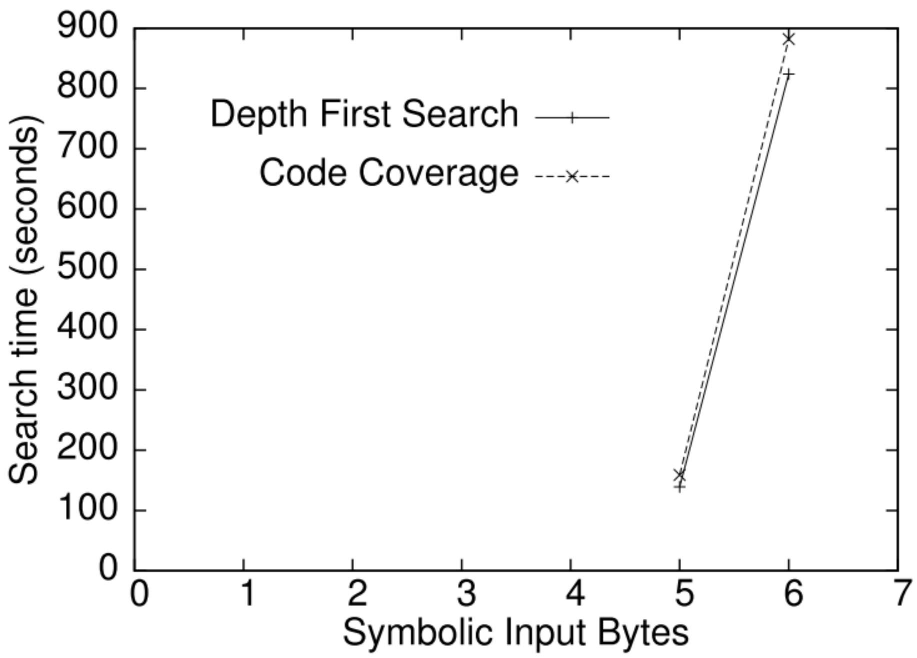
Only small part influences given loop

GET /long/path/file HTTP/1.1
Host: thisisthehost.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 1337

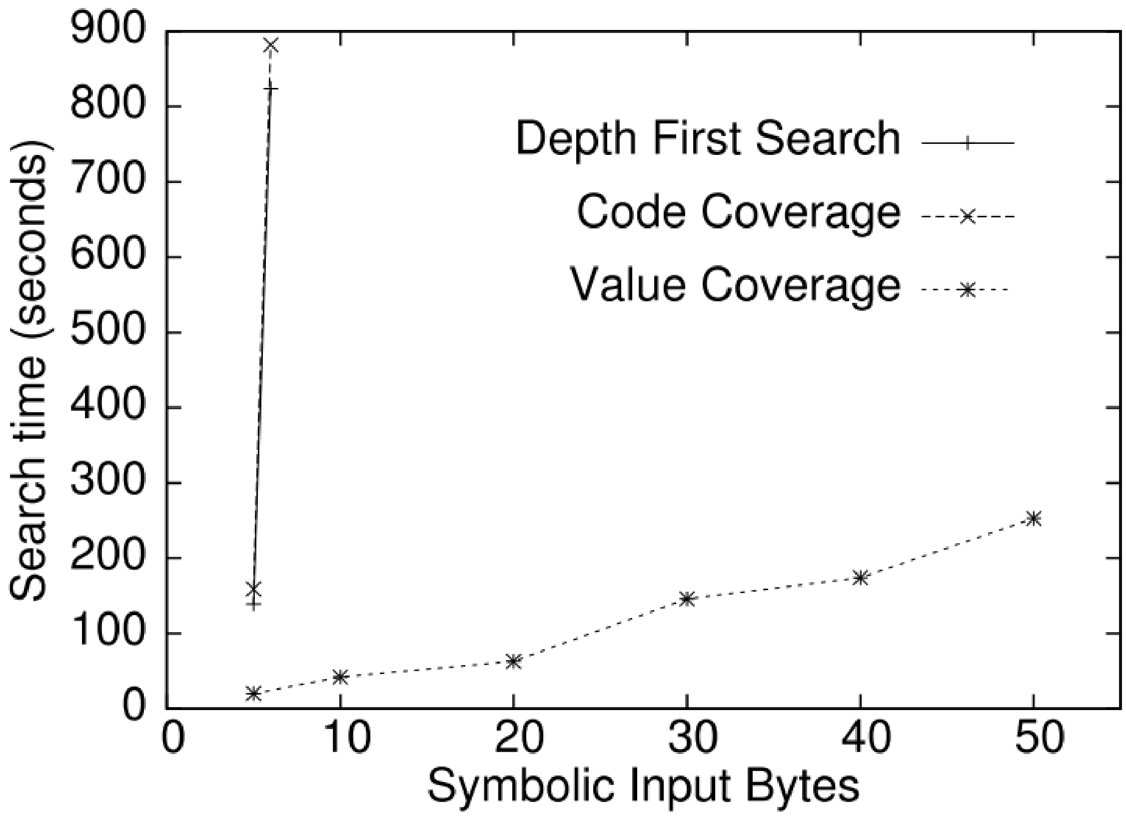
➔ Make only this part symbolic

+other clever tricks

Symbolic execution



Our approach



Results

Program	Vulnerability	Dowsing AG score	Loops	LoC	Symbolic input	Symbolic execution V-S2E	M-S2E	*Dowser*
nginx 0.6.32	CVE-2009-2629 heap underflow	4th out of 62/140 630 points	517	66k	URI field 50 bytes	> 8 h	> 8 h	253 sec
ffmpeg 0.5	UNKNOWN heap overread	3rd out of 727/1419 2186 points	1286	300k	Huffman table 224 bytes	> 8 h	> 8 h	48 sec
inspircd 1.1.22	CVE-2012-1836 heap overflow	1st out of 66/176 625 points	1750	45k	DNS response 301 bytes	200 sec	200 sec	32 sec
poppler 0.15.0	UNKNOWN heap overread	39th out of 388/904 1075 points	1737	120k	JPEG image 1024 bytes	> 8 h	> 8 h	14 sec
poppler 0.15.0	CVE-2010-3704 heap overflow	59th out of 388/904 910 points	1737	120k	Embedded font 1024 bytes	> 8 h	> 8 h	762 sec
libexif 0.6.20	CVE-2012-2841 heap overflow	8th out of 15/31 501 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	652 sec	652 sec
libexif 0.6.20	CVE-2012-2840 off-by-one error	15th out of 15/31 40 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	347 sec	347 sec
libexif 0.6.20	CVE-2012-2813 heap overflow	15th out of 15/31 40 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	277 sec	277 sec
snort 2.4.0	CVE-2005-3252 stack overflow	24th out of 60/174 246 points	616	75k	UDP packet 1100 bytes	> 8 h	> 8 h	617 sec

great stuff

Then we got the EUROSYS reviews…

- Overall merit:
  2. Top 50% but not top 25% of submitted papers
- Reviewer qualification:
  4. I know a lot about this area
- Strengths:
  – interesting set of heuristics for targeting buffer overflows
- Weaknesses:
  – the techniques are not clearly presented and justified
  – weak experimental evaluation, which provides little insight into the benefits of the different heuristics employed

Comments

Typical:

One contribution of the work is statically ranking array accesses based on a complexity metric. However, the authors don't present any data backing up the usefulness of that ranking. In particular, I would like to know whether there is any correlation between high-ranking and buggy memory accesses.

Comments

Typical:

Technique depends on concrete inputs executing array indexes. Starting from an execution "close" to the bug obviously makes a big difference. Comparing "pure" symbolic execution with their technique is unfair.

Comments

Typical:

Finding a single new bug is not a stellar result.

Comments

Typical:

related work: misses prior work on directed symbolic execution. For example, "predictive testing" [ESEC/FSE'07] "make zesti“ [ICSE'12].

Frankly,....

- The reviewers did an excellent job
- Very detailed
- Very thoughtful
- Very painful

(Overall score: 2, 3, 2, 4, 4 → reject)

Then comes the rebuttal

- Rebuttals are tricky
  - Often they make things worse for the author
- Three golden rules of rebuttals:
  1. do not promise to add what reviewer would like
  2. do not argue why it is not so bad
  3. stick to factual mistakes

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure" SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with “pure” SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, “Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel.”

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. “Is step 1 intra-procedural?”

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. “You need some knowledge of the input grammar for the field shifting optimization.”

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. “Need test suite that exercises vulnerable loop”

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. “Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure“ SE is unfair“

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. “The FSE'07 and ICSE'12 papers”

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. “SAGE has been successful in finding overflows”

All papers on Sage mention the ‘Generational search’ as the primary strategy guiding symbolic execution. [Long explanation.]

8. “Do the heuristics work?”

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?“

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, "Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel."

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. "Is step 1 intra-procedural?"

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. "You need some knowledge of the input grammar for the field shifting optimization."

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. "Need test suite that exercises vulnerable loop"

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. "Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure" SE is unfair"

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. "The FSE'07 and ICSE'12 papers"

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. "SAGE has been successful in finding overflows"

All papers on Sage mention the 'Generational search' as the primary strategy guiding symbolic execution. [Long explanation.]

8. "Do the heuristics work?"

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?"

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Thank you for the (excellent!) reviews. We have only a few clarifications:

1, "Using static analysis to find high-value targets, using DTA to find the right inputs, and guided symbolic execution to exploit the vuln. are not new, but the combination is novel."

We agree that static analysis, DTA and symbolic execution (and even combinations thereof) are nothing new, but believe our work is more than just a combination of existing ideas. [blah-blah-blah].

2. "Is step 1 intra-procedural?"

Yes. We currently only employ intra-procedural analysis, but the heuristic itself is independent of the way the dataflow graph is generated.

3. "You need some knowledge of the input grammar for the field shifting optimization."

This is true. Fortunately, such knowledge is available for many applications (certainly when vendors test their own code). We do not need full knowledge of the input grammar. For instance, we need not understand the contents or effects of fields.

4. "Need test suite that exercises vulnerable loop"

True. The problem of code coverage exists for dynamic analysis in general. Several SE projects explicitly address the problem of code coverage and we could use them for our work.

5. "Since the technique depends on concrete inputs executing array indexes, starting from an execution "close" to the bug obviously makes a big difference. Comparing with "pure" SE is unfair"

Pure symbolic execution is also applied using the concrete input as starting point, so there is no unfairness in the evaluation. We never just run symbolic execution without any starting input.

6. "The FSE'07 and ICSE'12 papers"

These papers are truly relevant in that they employ test cases as input seeds for a symbolic search towards buffer overflows. However, we feel they are complementary to our work, since blah-blah-blah

7. "SAGE has been successful in finding overflows"

All papers on Sage mention the 'Generational search' as the primary strategy guiding symbolic execution. [Long explanation.]

8. "Do the heuristics work?"

We believe they do in the sense that we found very complicated and real bugs with them. [blah-blah-blah]

9. "How are the short symbolic inputs constructed?"

In the same way as in the regular 'magic' inputs for arrays - only the first bytes are made symbolic, the rest remains concrete.

Reject

The paper was discussed at the PC meeting, but not accepted. PC agreed that the combination of techniques used was novel. The main concerns were the detail of the exploration of the heuristics (e.g., contribution of different techniques to the overall results, and the sensitivity to the choice of numeric parameters), and the question of whether or not the techniques would be effective on new workloads which had not been used while developing the system.

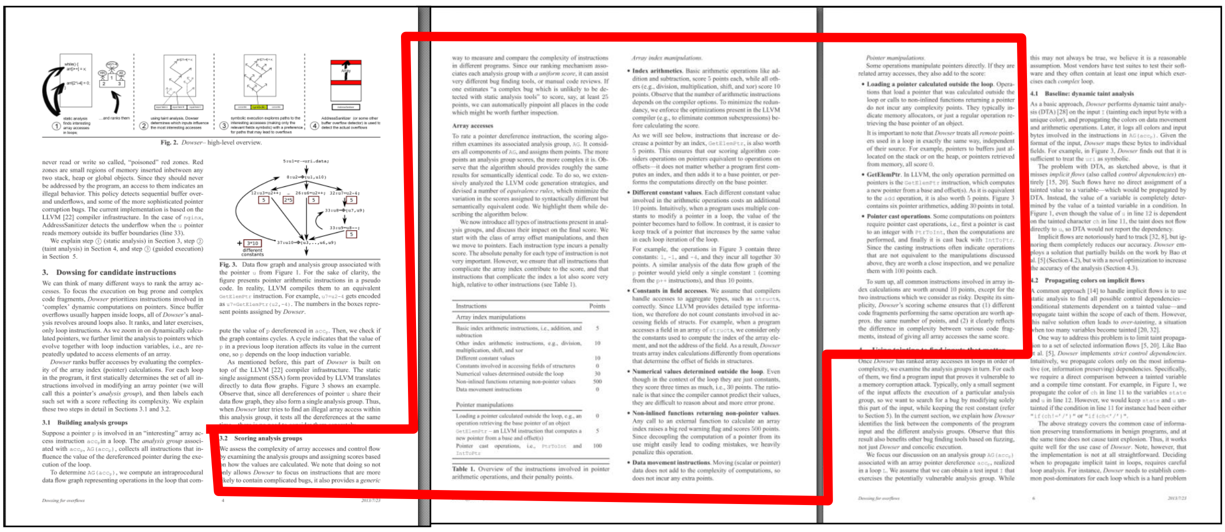
How to proceed?

- Filter the criticism
  - Focus on what is important
  - In our case: the heuristics

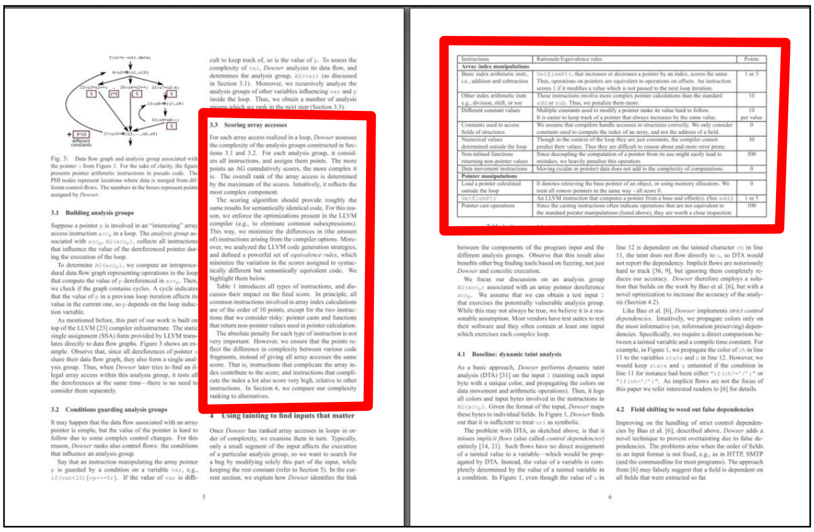
Strategy

- Shrink section explaining our heuristics
- Evaluate the heuristics

Old



New



Program	Vulnerability	Dowsing AG score	Loops	LoC	Symbolic input	Symbolic execution V-S2E	M-S2E	*Dowser*
nginx 0.6.32	CVE-2009-2629 heap underflow	4th out of 62/140 630 points	517	66k	URI field 50 bytes	> 8 h	> 8 h	253 sec
ffmpeg 0.5	UNKNOWN heap overread	3rd out of 727/1419 2186 points	1286	300k	Huffman table 224 bytes	> 8 h	> 8 h	48 sec
inspircd 1.1.22	CVE-2012-1836 heap overflow	1st out of 66/176 625 points	1750	45k	DNS response 301 bytes	200 sec	200 sec	32 sec
poppler 0.15.0	UNKNOWN heap overread	39th out of 388/904 1075 points	1737	120k	JPEG image 1024 bytes	> 8 h	> 8 h	14 sec
poppler 0.15.0	CVE-2010-3704 heap overflow	59th out of 388/904 910 points	1737	120k	Embedded font 1024 bytes	> 8 h	> 8 h	762 sec
libexif 0.6.20	CVE-2012-2841 heap overflow	8th out of 15/31 501 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	652 sec	652 sec
libexif 0.6.20	CVE-2012-2840 off-by-one error	15th out of 15/31 40 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	347 sec	347 sec
libexif 0.6.20	CVE-2012-2813 heap overflow	15th out of 15/31 40 points	121	10k	EXIF tag/length 1024 + 4 bytes	> 8 h	277 sec	277 sec
snort 2.4.0	CVE-2005-3252 stack overflow	24th out of 60/174 246 points	616	75k	UDP packet 1100 bytes	> 8 h	> 8 h	617 sec

Table 2: Applications tested with *Dowser*. The *Dowsing* section presents the results of *Dowser*'s ranking scheme. *AG score* is the complexity of the vulnerable analysis group - its position among other analysis groups; X/Y denotes all analysis groups that are "complex enough" to be potentially analyzed/all analysis groups which access arrays; and the number of points it scores. *Loops* counts outermost loops in the whole program, and *LoC* - the lines of code according to `sloccount`. *Symbolic input* specifies how many and which parts of the input were determined to be marked as symbolic by the first two components of *Dowser*. The last section shows symbolic execution times until revealing the bug. Almost all applications proved to be too complex for the vanilla version of S2E (*V-S2E*). *Magic S2E* (*M-S2E*) is the time S2E takes to find the bug when we feed it with an input with only a minimal symbolic part (as identified in Symbolic input). Finally, the last column is the execution time of fully-fledged *Dowser*.

`inspircd`, `libexif`, `poppler`, and `snort`. Additionally, we consider the vulnerabilities in `sendmail` tested by Zitser et al. [45]. For these applications, we analyzed all buffer overflows reported in CVE [26] since 2009. For `ffmpeg`, rather than include all possible codecs, we just picked the ones for which we had test cases. Out of 27 CVE reports, we took 17 for the evaluation. The remaining ten vulnerabilities are out of the scope of this paper – nine of them are related to an erroneous usage of a correct function, e.g., `strcpy`, and one was not in a loop. In this section, we consider the analysis groups from all the applications together, giving us over 3000 samples, 17 of which are known to be vulnerable[4].

When evaluating *Dowser*'s scoring mechanism, we also compare it to a straightforward scoring function that treats all instructions uniformly. For each array access, it considers exactly the same AGs as *Dowser*. However, instead of the scoring algorithm (Table 1), each instruction gets 10 points. We will refer to this metric as `count`.

Correlation For both *Dowser*'s and the `count` scoring functions, we computed the correlation between the number of points assigned to an analysis group and the existence of a memory corruption vulnerability. We used the Spearman rank correlation [2], since it is a reliable measure that is appropriate even when we do not know the probability distribution of the variables, or when the association between the variables is non-linear.

The positive correlation for *Dowser* is statistically significant at $p < 0.0001$, for `count` — at $p < 0.005$. The correlation for *Dowser* is stronger.

Dowsing The *Dowsing* columns of Table 2 shows that our focus on complex loops limits the search space from thousands of LoC to hundreds of loops, and finally to a small number of "interesting" analysis groups. Observe that `ffmpeg` has more analysis groups than loops. That is correct. If a loop accesses multiple arrays, it contains multiple analysis groups.

By limiting the analysis to complex cases, we focus on a smaller fraction of all AGs in the program, e.g., we consider 36.9% of all the analysis groups in `inspircd`, and 34.5% in `snort`. `ffmpeg`, on the other hand, contains lots of complex loops that decode videos, so we also observe many "complex" analysis groups.

In practice, symbolic execution, guided or not is expensive, and we can hardly afford a thorough analysis of more than just a small fraction of the target AGs of an application, say 20%-30%. For this reason, *Dowser* uses a scoring function, and tests the analysis groups in order of

[4]Since the scoring functions are application agnostic, it is sound to compare their results across applications.

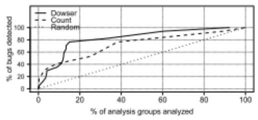


Fig. 6: A comparison of random testing and two scoring functions: *Dowser*'s and `count`. It illustrates how many bugs we detect if we test a particular fraction of the analysis groups.

decreasing score. Specifically, *Dowser* looks at complexity. However, alternative heuristics are also possible. For instance, one may count the instructions that influence array accesses in an AG. To evaluate whether *Dowser*'s heuristics are useful, we compare how many bugs we discover if we examine increasing fractions of all AGs, in descending order of the score. So, we determine how many of the bugs we find if we explore the top 10% of all AGs, how many bugs we find when we explore the top 20%, and so on. In our evaluation, we are comparing the following ranking functions: (1) *Dowser*'s complexity metric, (2) counting instructions as described above, and (3) random.

Figure 6 illustrates the results. The random ranking serves as a baseline—clearly both `count` and *Dowser* perform better. In order to detect all 17 bugs, *Dowser* has to analyze 92.2% of all the analysis groups. However, even with just 15% of the targets, we find almost 80% (13/17) of all the bugs. At that same fraction of targets, `count` finds a little over 40% of the bugs (7/17). Overall, *Dowser* outperforms `count` beyond the 10% in the ranking. It also reaches the 100% bug score earlier than the alternatives, although the difference is minimal.

The reason why *Dowser* still requires 92% of the AGs to find *all* bugs, is that some of the bugs were very simple. The "simplest" cases include a trivial buffer overflow in `poppler` (worth 16 points), and two vulnerabilities in `sendmail` from 1999 (worth 20 points each). Since *Dowser* is designed to prioritize complex array accesses, these buffer overflows end up in the low scoring group. (The "simple" analysis groups – with less than 26 points – start at 47.9%). Clearly, both heuristics provide much better results than random sampling. Except for the tail, they find the bugs significantly quicker, which proves their usefulness.

To summarize, we have shown that a testing strategy based on *Dowser*'s scoring function is effective. It lets us find vulnerabilities quicker than random testing or a scoring function based on the length of an analysis group.

6.2.2 Symbolic execution

Table 2 presents attacks detected by *Dowser*. The last section shows how long it takes before symbolic execution detects the bug. Since the vanilla version of S2E cannot handle these applications with the whole input marked as symbolic, we also run the experiments with minimal symbolic inputs ("*Magic* S2E"). It represents the best-case scenario when an all-knowing oracle tells the execution engine exactly which bytes it should make symbolic. Finally, we present *Dowser*'s execution times.

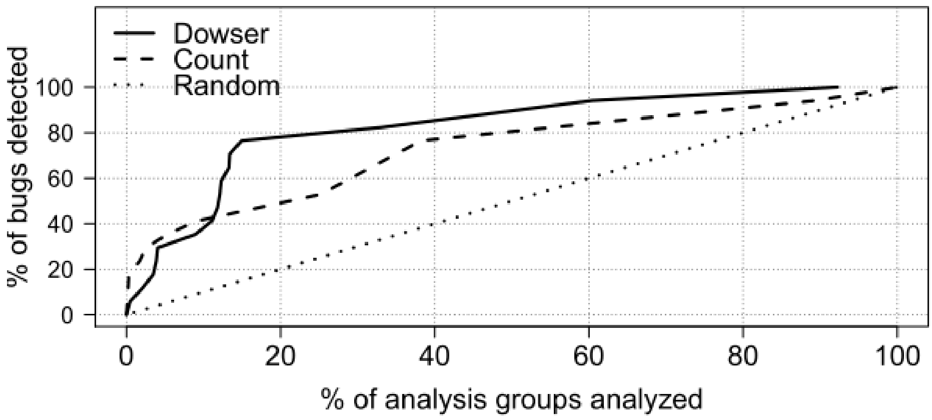
We run S2E for as short a time as possible, e.g., a single request/response in `nginx` and transcoding a single frame in `ffmpeg`. Still, in most applications, vanilla S2E fails to find bugs in a reasonable amount of time. `inspircd` is an exception, but in this case we explicitly tested the vulnerable DNS resolver only. In the case of `libexif`, we can see no difference between "Magic S2E" and *Dowser*, so *Dowser*'s guidance did not influence the results. The reason is that our test suite here was simple, and the execution paths reached the vulnerability condition quickly. In contrast, more complex applications process the inputs intensively, moving symbolic execution away from the code of interest. In all these cases, *Dowser* finds bugs significantly faster. Even if we take the 15 minute tests of higher-ranking analysis groups into account, *Dowser* provides a considerable improvement over existing systems.

7 Related work

Dowser is a 'guided' fuzzer which draws on knowledge from multiple domains. In this section, we place our system in the context of existing approaches. We start with the scoring function and selection of code fragments. Next, we discuss traditional fuzzing. We then review previous work on dynamic taint analysis in fuzzing, and finally, discuss existing work on whitebox fuzzing and symbolic execution.

Software complexity metrics Many studies have shown that software complexity metrics are positively correlated with defect density or security vulnerabilities [29, 35, 16, 44, 35, 32]. However, Nagappan et al. [29] argued that no single set of metrics fits all projects, while Zimmermann et al. [44] emphasize a need for metrics that exploit the unique characteristics of vulnerabilities, e.g., buffer overflows or integer overruns. All these approaches consider the broad class of post-release defects or security vulnerabilities, and consider a very generic set of measurements, e.g., the number of basic blocks in a function's control flow graph, the number of global or local variables read or written, the maximum nesting level

Evaluated heuristics



+

The positive correlation for *Dowser* is statistically significant at $p < 0.0001$, for `count` — at $p < 0.005$. The correlation for *Dowser* is stronger.

Better related work

\+ Better explanation

More applications

= much better paper

USENIX SEC

Finally: important lesson for students

- Even though
  - someone is an insensitive jerk
  - with a personal vendetta against your advisor,
  - no concern for human dignity and feelings,
  - Acting with a primary agenda of promoting their own greatness,

  they still often have intellectually useful suggestions.

2.3 Session 3: Best papers from the EU projects

This showcase session was dedicated to recognizing the contributions of the European Commission and the Seventh Framework Programme, by presenting excellent research by EU-funded projects to our students.

2.3.1 Eradicating DNS Rebinding with the Extended Same-Origin Policy

EU Project Websand.

Authors Sebastian Lekies, Ben Stock, Martin Johns.

Speaker Sebastian Lekies.

Paper Summary The Web's principal security policy is the Same-Origin Policy (SOP), which enforces origin-based isolation of mutually distrusting Web applications. Since the early days, the SOP was repeatedly undermined with variants of the DNS Rebinding attack, allowing untrusted script code to gain illegitimate access to protected network resources. To counter these attacks, the browser vendors introduced countermeasures, such as DNS Pinning, to mitigate the attack. In this paper, we present a novel DNS Rebinding attack method leveraging the HTML5 Application Cache. Our attack allows reliable DNS Rebinding attacks, circumventing all currently deployed browser-based defense measures. Furthermore, we analyze the fundamental problem which allows DNS Rebinding to work in the first place: The SOP's main purpose is to ensure security boundaries of Web servers. However, the Web servers themselves are only indirectly involved in the corresponding security decision. Instead, the SOP relies on information obtained from the domain name system, which is not necessarily controlled by the Web server's owners. This mismatch is exploited by DNS Rebinding. Based on this insight, we propose a light-weight extension to the SOP which takes Web server provided information into account. We successfully implemented our extended SOP for the Chromium Web browser and report on our implementation's interoperability and security properties.

Eradicating DNS Rebinding with the Extended Same-Origin Policy

Sebastian Lekies
July 24th, 2013

Agenda

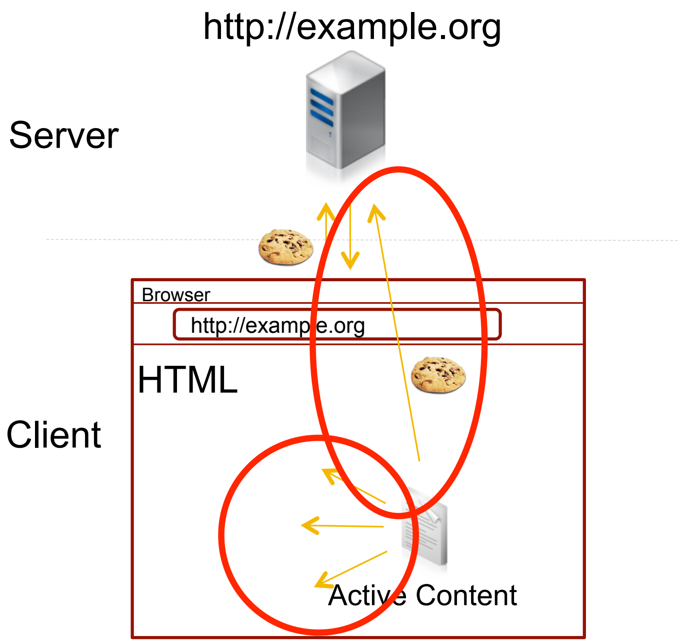
Technical Background
- Web application 101
- The Same-Origin Policy

DNS Rebinding
- The basic attack
- History repeating

Extending the Same-Origin Policy
- The three principals of Web interaction
- Extending the SOP with server-provided information

Technical Background
Web Application 101

Web Application Paradigm



Active Content enables Web Apps to…

- …interact with the Document (via the DOM)
- …interact with the Server (via XMLHttpRequest, iFrames, etc)

…in the name of the user

- security sensitive (!)
- sensitive data and active content can originate from different origins
- access is governed by the Same-Origin Policy

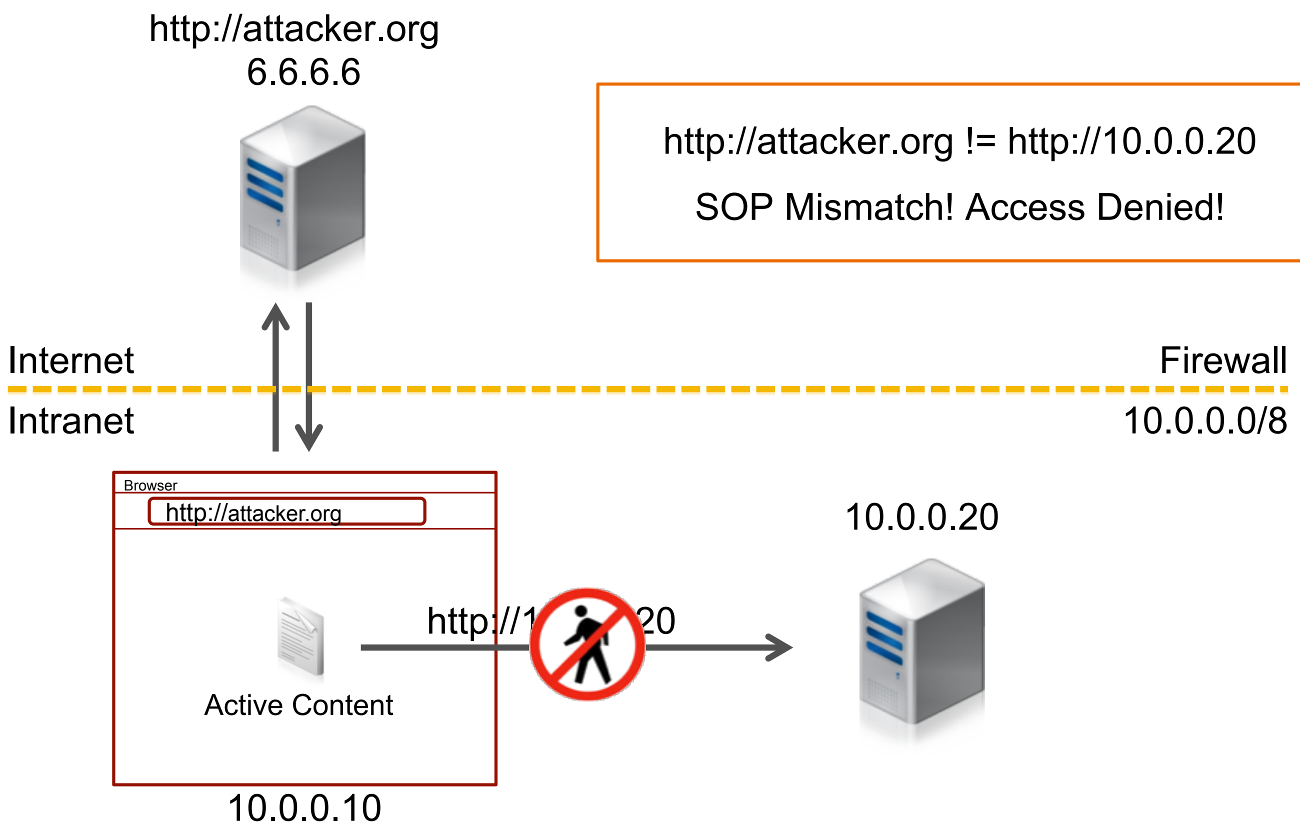
Technical Background
The Same-Origin Policy (SOP)

> The Same-Origin Policy **restricts access** of active content to objects that share the same origin. The origin is, hereby, defined by the **protocol**, the **domain** and the **port** used to retrieve the object.

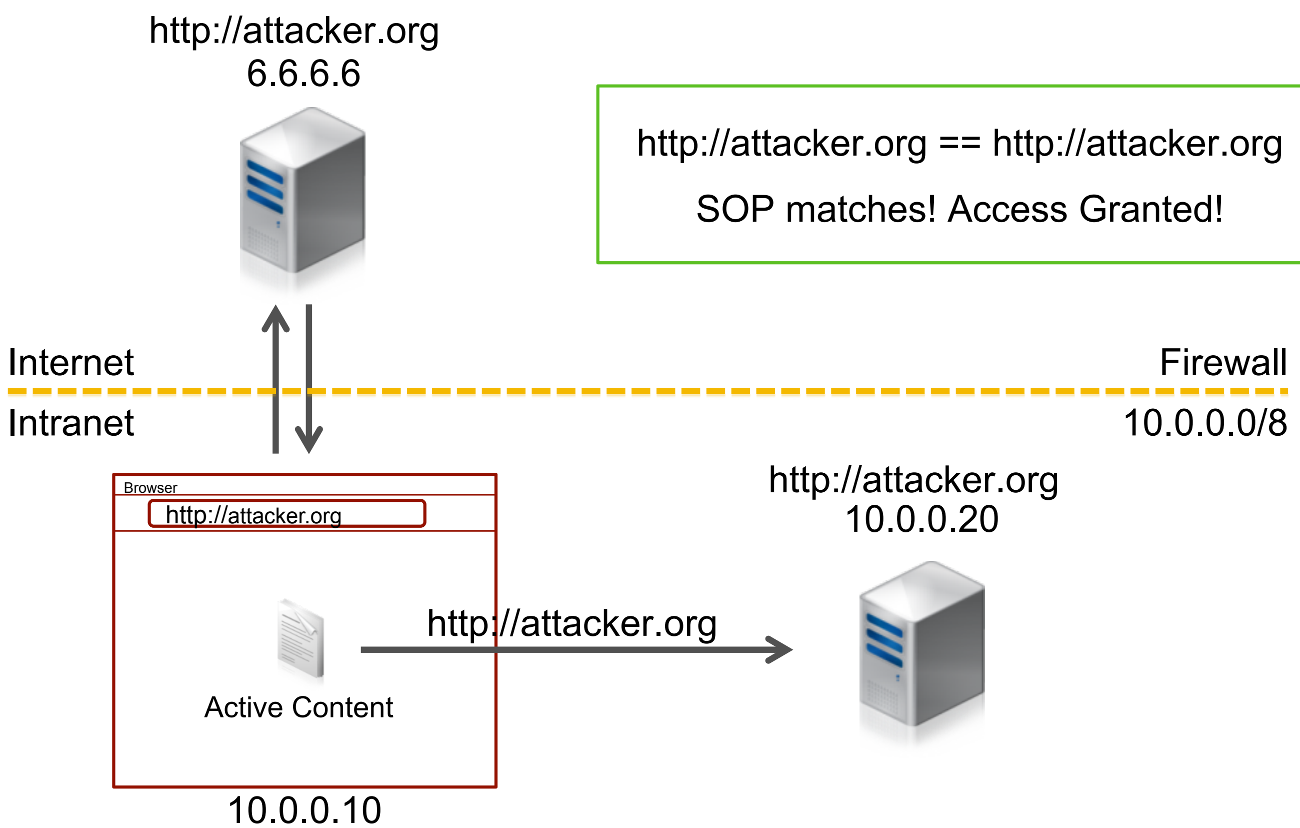
$\underbrace{\text{http://}}_{\text{protocol}}\underbrace{\text{example.org}}_{\text{domain}}:\underbrace{\text{80}}_{\text{port}}\text{/some/webpage.html}$

Target host	Access	Reason
http://example.org	Yes	---
https://example.org	No	Protocol mismatch
http://example.org:8080	No	Port mismatch
http://facebook.com	No	Domain mismatch

The Same-Origin Policy
Protecting the Intranet



DNS-Rebinding
The basic attack



DNS-Rebinding
History repeating

Attack:

1996: The Princeton Attack

Java

- In 1996 Java applets offered sophisticated networking capabilities
- DNS-server returned two IP addresses for the same host
  1. The IP the applet was loaded from
  2. The IP of the target host

Countermeasures:

Strict IP-based access control for Java applets

- Java applets are only allowed to connect to its server's IP address
- Maintained over the entire lifetime of the applet
  - Including a Browser's Java Cache

DNS-Rebinding
History repeating

Attack:

2002: JavaScript

Java Script

- DNS-Rebinding via domain relaxation
  - Domain 1: attacker.org → 10.0.0.20
  - Domain 2: evil.attacker.org → 6.6.6.6
- Quick-swap DNS

Countermeasures:

Explicit domain relaxation

- A domain has to explicitly grant access via domain relaxation

DNS-Pinning

- The browser caches the DNS-to-IP mapping
- The browser resolves the mapping only once

DNS-Rebinding
History repeating

Attack:

Java Script

2006: The full browser experience

- Martin Johns discovered a way to drop DNS-to-IP-mapping (FF & IE)
- Leading to many DNS-rebinding vulnerabilities in…
  - …JavaScript, Flash, Java
  - Even allowing socket-communication

Countermeasures:

Host-Header checking

- In HTTP 1.1 a browser attaches an additional header field containing the host
- Applications need to check this header for correctness

Restrictive Networking Capabilities for browser plug-ins

- Plug-ins are only allowed to connect to a limited set of ports.

DNS-Rebinding
History repeating

Attack:

2013: HTML5 Offline Application Cache

- DNS-pinning can only be maintained for a short amount of time
- HTML5 AppCache enables a…
  - …controllable caching behavior
  - …a way for content to easily exceed DNS pinning times

Countermeasure:

???

Extending the Same-Origin Policy
The three principals of Web interaction

The Same-Origin Policy's duty is…
- …to isolate unrelated Web applications from each other…
- …based on the origin of the interacting resources

The semantics of the SOP are built around two entities
1. The ***browser*** enforces the policy
2. The ***server*** provides the resources which are the subject of the policy decision

However, the entities involved in the implementation of the SOP differ
1. The ***browser*** enforces the policy
2. The ***network*** (DNS-System) provides the underlying information

The server is not involved in the policy decision (!)
- Hence, the ***network*** governs the ***server's*** security characteristics

Extending the Same-Origin Policy
Extending the SOP with server-provided information

Only the server should be capable of setting its trust boundary
- Currently, the browser is guessing this boundary…
- …based on information delivered by the network

Therefore, we propose to extend the Same-Origin policy:
- With server-provided input
- Delivered through an HTTP response header

{ protocol, domain, port, *server-origin* }

A server's trust boundary could comprise multiple domains:
- E.g. www.example.org, example.org, example.net
- The server's origin is, therefore, a comma-separated list of domain names

Extending the Same-Origin Policy
eSOP decision Logic

The eSOP is satisfied iff:

$$\{\textbf{protocol}, \textbf{domain}, \textbf{port}\}_A == \{\textbf{protocol}, \textbf{domain}, \textbf{port}\}_T$$

and

$$\textbf{domain}_A \in \textbf{server-origin}_T$$

If the **server-origin**$_T$ property is empty, the second criterion always evaluates as "true".

Example

- 10.0.0.20's server-origin = { 10.0.0.20, wiki.corp }
- 2. part of the SOP decision: attacker.org ∈ of { 10.0.0.20, wiki.corp } → false
- Many edge cases are explained in the paper

Conclusion

The Same-Origin Policy is the most basic security policy in modern browsers

- It isolates unrelated Web applications from each other…
- …based on the origin of the interacting resources (protocol, domain, port)

DNS-Rebinding circumvents the SOP…

- …by associating a DNS-name with two unrelated IPs
- Major vulnerabilities have been discovered in 1996, 2002, 2006, 2013

DNS-Rebinding is a protocol-level flaw

- The network governs the server's security characteristics
- We enhanced the SOP with explicit server-origin to eradicate DNS-rebinding

We implemented our approach within Chromium

Thank you

Contact information:

Sebastian Lekies
@sebastianlekies
Sebatian.Lekies@sap.com

2.3.2 Specialization and Outsourcing in the Malware Ecosystem

EU Project NESSOS.

Speaker Juan Caballero.

Talk Summary In the cybercrime ecosystem attackers have understood that tackling the entire monetization chain is a daunting task requiring highly developed skills and resources. Thus, specialized services have emerged to outsource key parts to third parties such as malware toolkits, exploit marketplaces, and pay-per-install services. Such outsourcing encourages innovation and specialization, enabling attackers to focus on their end goals. This talk describes different components of this complex ecosystem, highlights key research issues, and discusses operational implications.

www.software.imdea.org

Specialization in the Malware Distribution Ecosystem

Juan Caballero (IMDEA Software Institute, Madrid)
July 24th, 2013
Bochum

www.software.imdea.org

Cybercrime Motivation

Malware in Cybercrime

- Internet-connected computers are worth money
- Malware used to monetize them

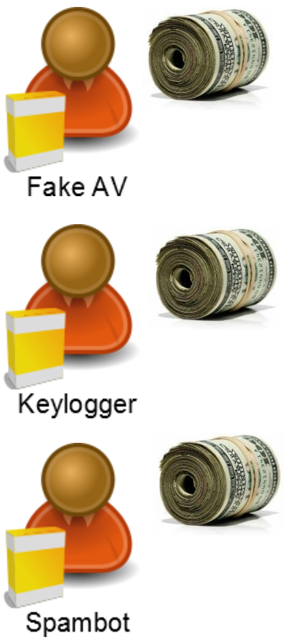
Monetizing the Malware

Malware for Dummies

www.software.imdea.org

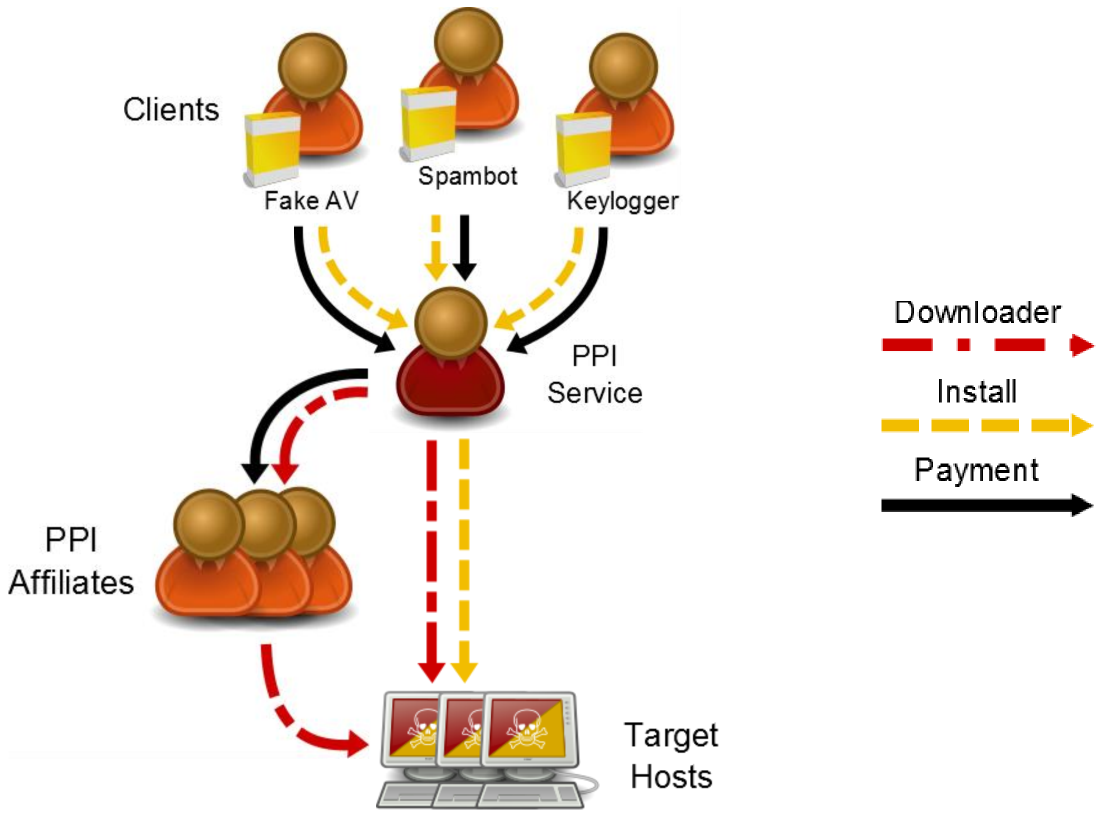
Malware Distribution

Malware Distribution: Outsourcing



Pay-per-Install
Exploitation-as-a-Service
Exploit Kits

Pay-Per-Install (PPI)



www.software.imdea.org

PPI: Prices Paid to Affiliates

Country	Price
OTH	13$
US	150$
GB	110$
CA	110$
DE	30$
BE	20$
IT	65$
CH	20$
CZ	20$

institute imdea software
www.software.imdea.org

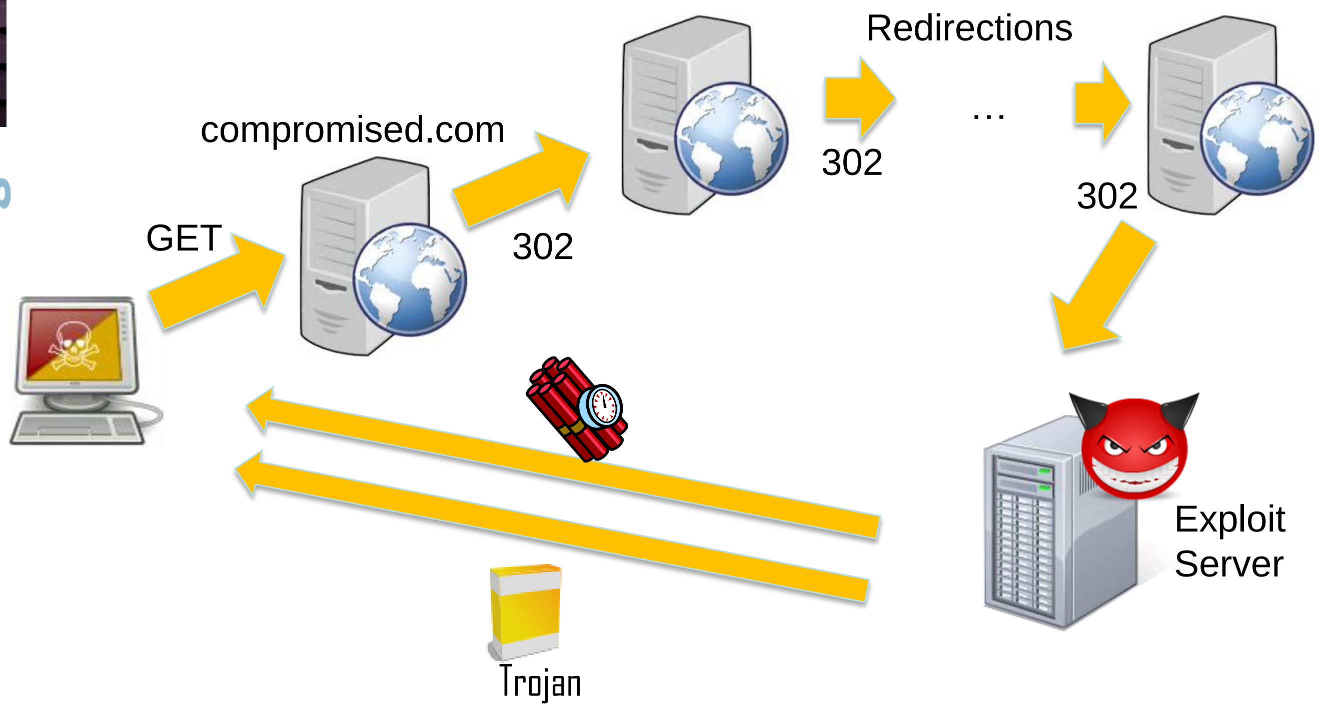
PPI: Pros & Cons

- ✓ Decouples compromise & monetization
- ✓ Investment reduction
- ✓ Access to multiple distribution vectors
- ✓ Independent innovation

- ✗ Lack of control
- ✗ Multiple installs on same host
- ✗ Shaving to affiliates
- ✗ Affiliates work with multiple programs

Alternative Web exploit services

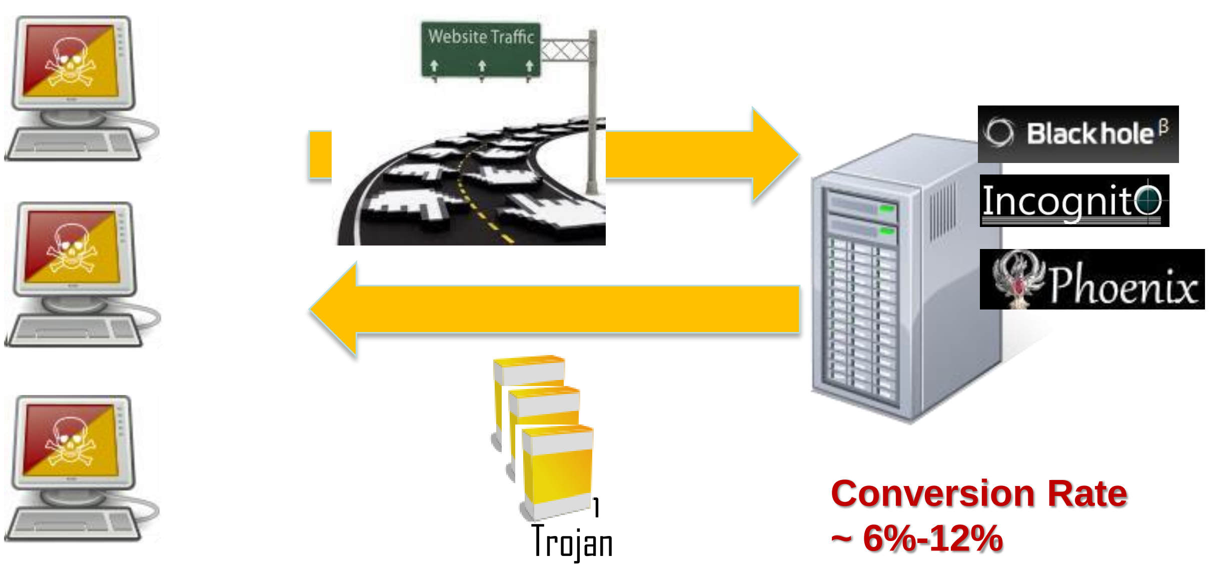
Drive-by Download



institute imdea software
www.software.imdea.org

Drive-by Download: Intuition

Converts Traffic into Installs



Conversion Rate ~ 6%-12%

Drive-by Outsourcing

- 3 things needed for drive-by download:

1. Software
2. Exploit Server (HW + Hosting)
3. Traffic

Exploit Kit

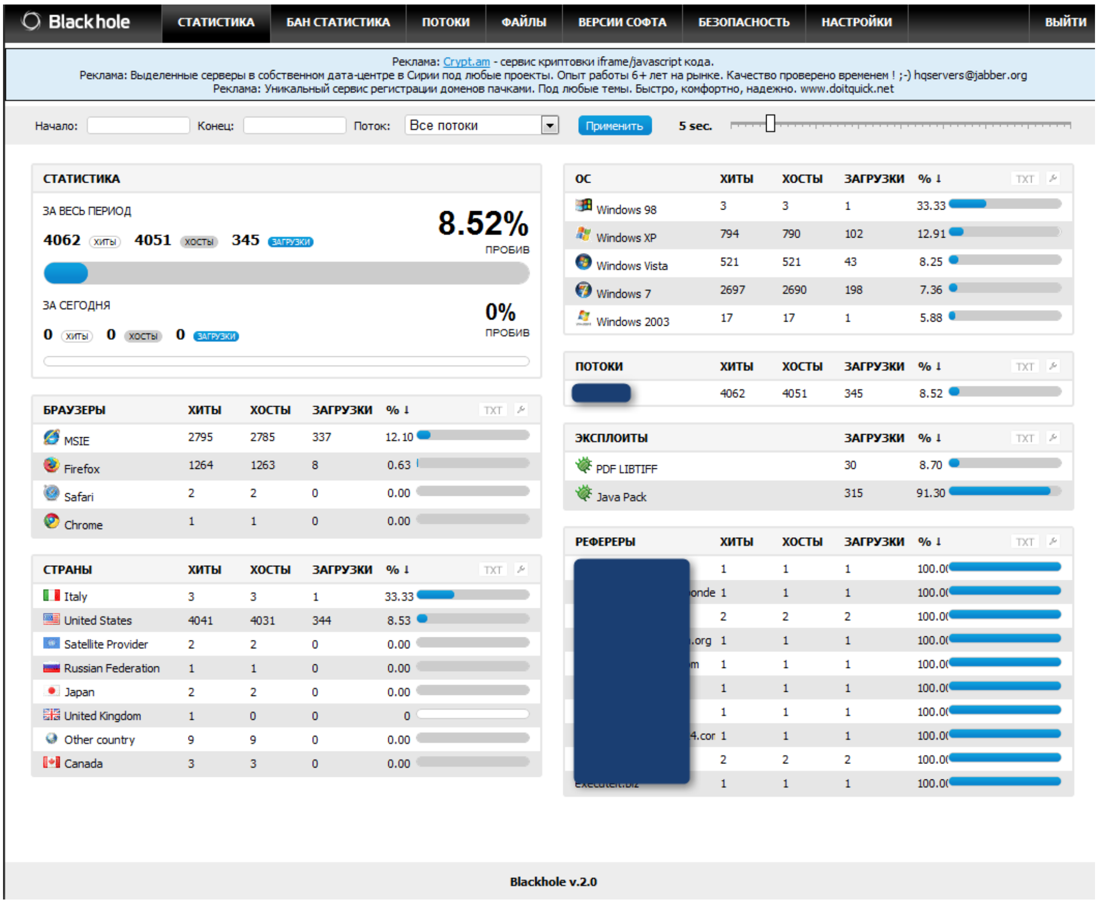
Exploitation-as-a-service

Pay-per-install

Exploit Kits

- Bundles exploits
  - Browser, Flash, Java
- Installs on web server
  - Add PHP code to site
- Configuration interface
  - Files, Referers, …

BlackHole 2.0 (2012)



Exploit Kits: Licensing

Fake AV

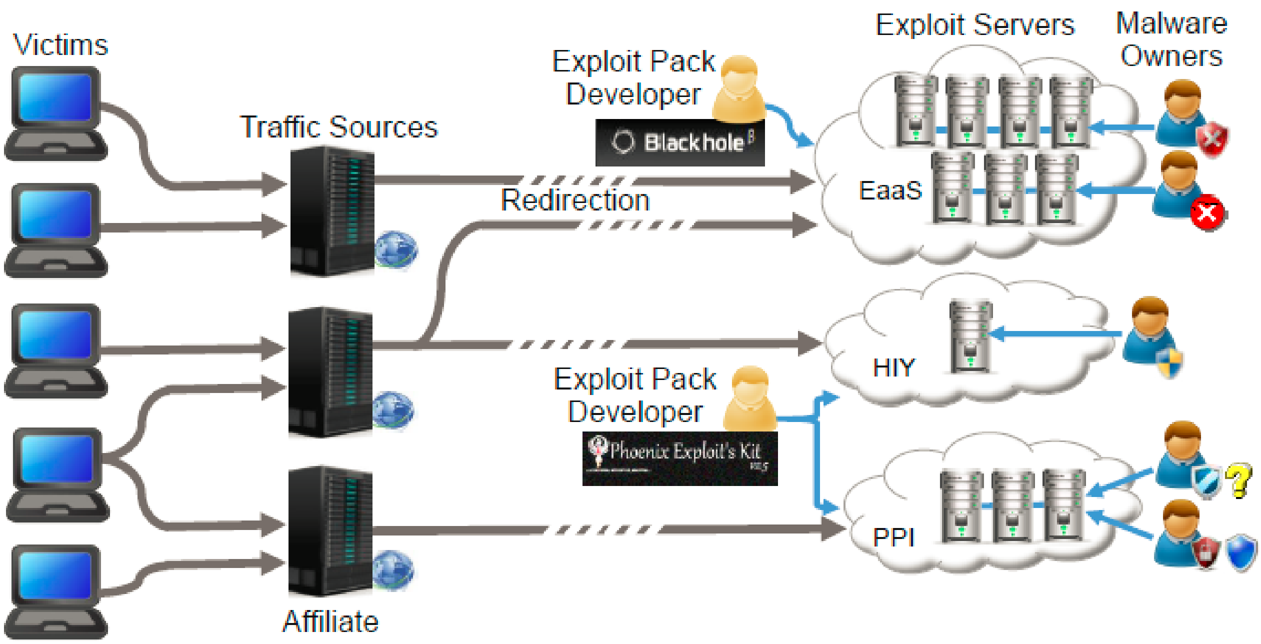
- Licenses
  1. One time fee (Phoenix)
     - $400 (2009)
     - $2200 (2011)
  2. Time-limited access
     - Free exploit updates
- Single or Multi-domain

- Server
- Domain
- Traffic

Exploitation-as-a-Service (EaaS)

- Rent a exploit server
  - Exploit kit license included
  - Configure through web interface
  - Diversity: ISP, geographical
- BlackHole
  - $50 / week, $500 / month
  - Single domain or multi-domain
- Other Models
  - Pay with part of your traffic

Drive-by-Download Ecosystem



Our Contributions

- Analysis of PPI (Usenix Security 2011)

Joint work with C. Grier, C. Kreibich & V. Paxson

- Analysis of EaaS (CCS 2012)

Joint work with C. Grier et al.

- Analysis of Drive-by Operations & Abuse Reporting (DIMVA 2013)

Joint work with A. Nappa & M. Z. Rafique

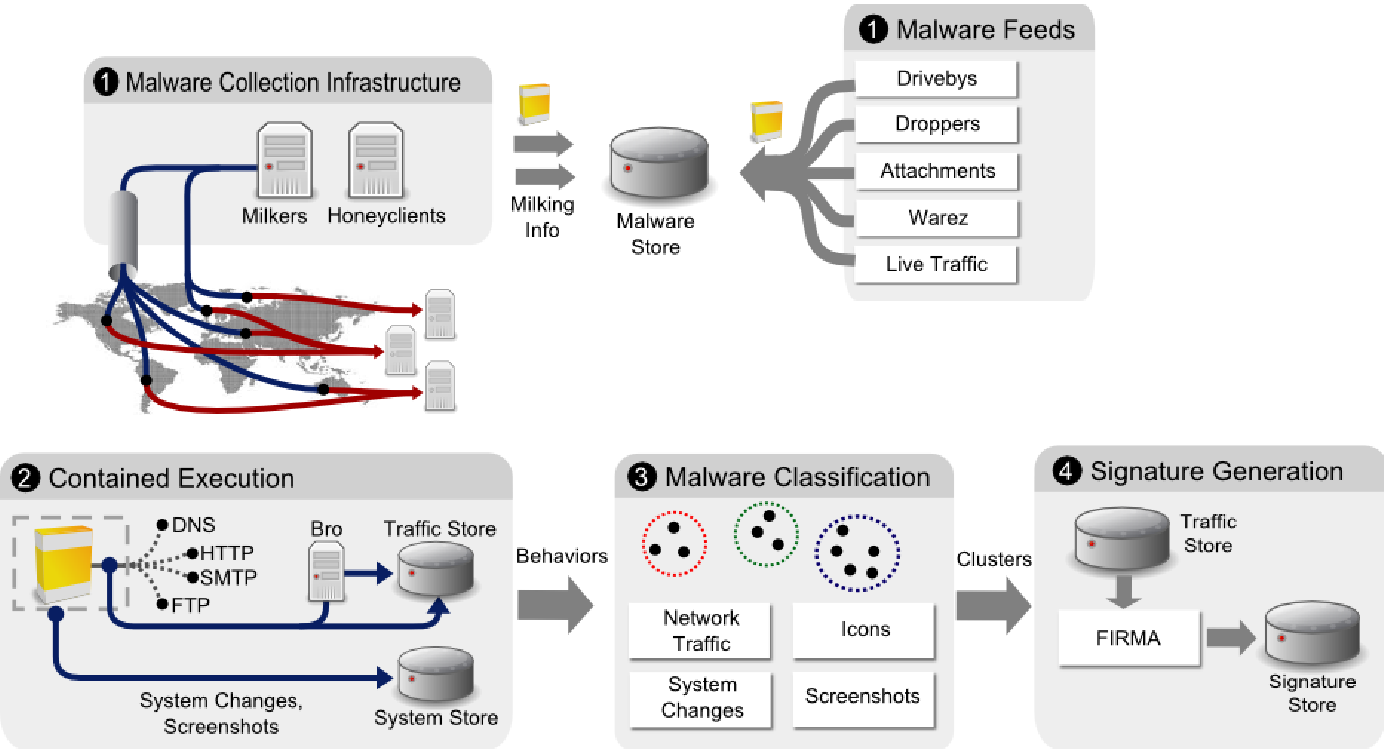
Outline

Intro

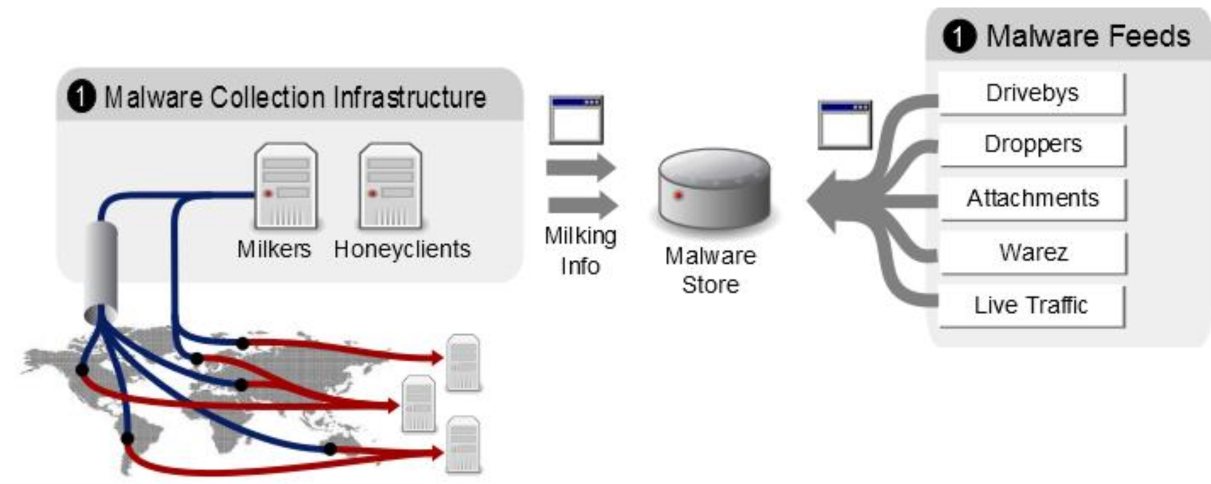
Architecture

Selected Results

Architecture Overview



Malware Collection



- Milkers & Honeyclients
  - Periodic
  - Anonimity & Geographical diversity
- External Malware Feeds

Malware Collected

Low feed overlap: 0.3 - 0.4%

Milkers	Vector	Start	End	# Downloads	# Malware
LoaderAdv	PPI	08/2010	02/2011	696,714	4,334
GoldInstall	PPI	08/2010	02/2011	361,325	4,488
Virut	PPI	08/2010	02/2011	4,841	72
Zlob	PPI	01/2011	02/2011	504	259

http://malicia-project.com

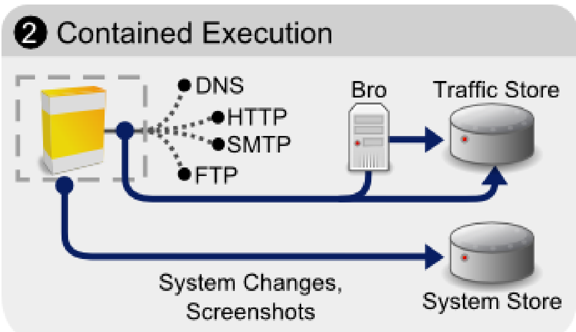
Honeyclients	Vector	Start	End	Malware	Servers
MALICIA	Drive-by	4/2012	3/2013	11,688	500

NEW

Feeds	Vector	Start	End	Malware
Google	Drive-by	4/2012	5/2012	4,967
Sandnet	Dropper	9/2011	5/2012	2,619
Spam Traps	Attachment	2/2012	5/2012	2,817
Torrents	Warez	9/2011	5/2012	17,182
Arbor	Mix	8/2011	5/2012	28,300

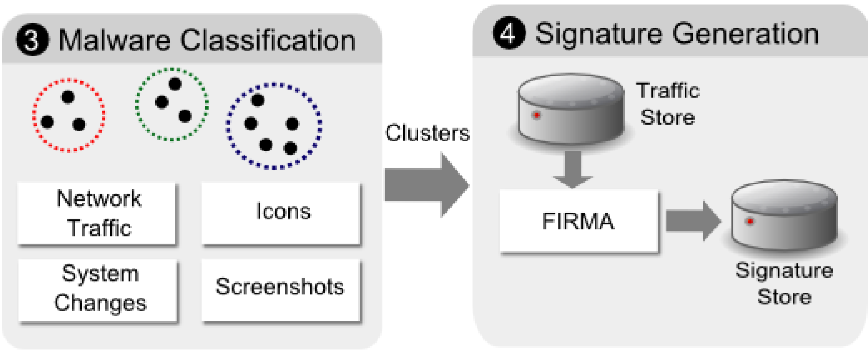
www.software.imdea.org

Malware Execution



- Contained environment
  - Mediated Internet connectivity
- Captures:
  - Network traffic
  - Screenshots
  - System changes

Malware Classification



1. Cluster malware
2. Label clusters with family names
3. Generate signatures
4. Analyze family monetization

Outline

Intro

Architecture

Selected Results

www.software.imdea.org

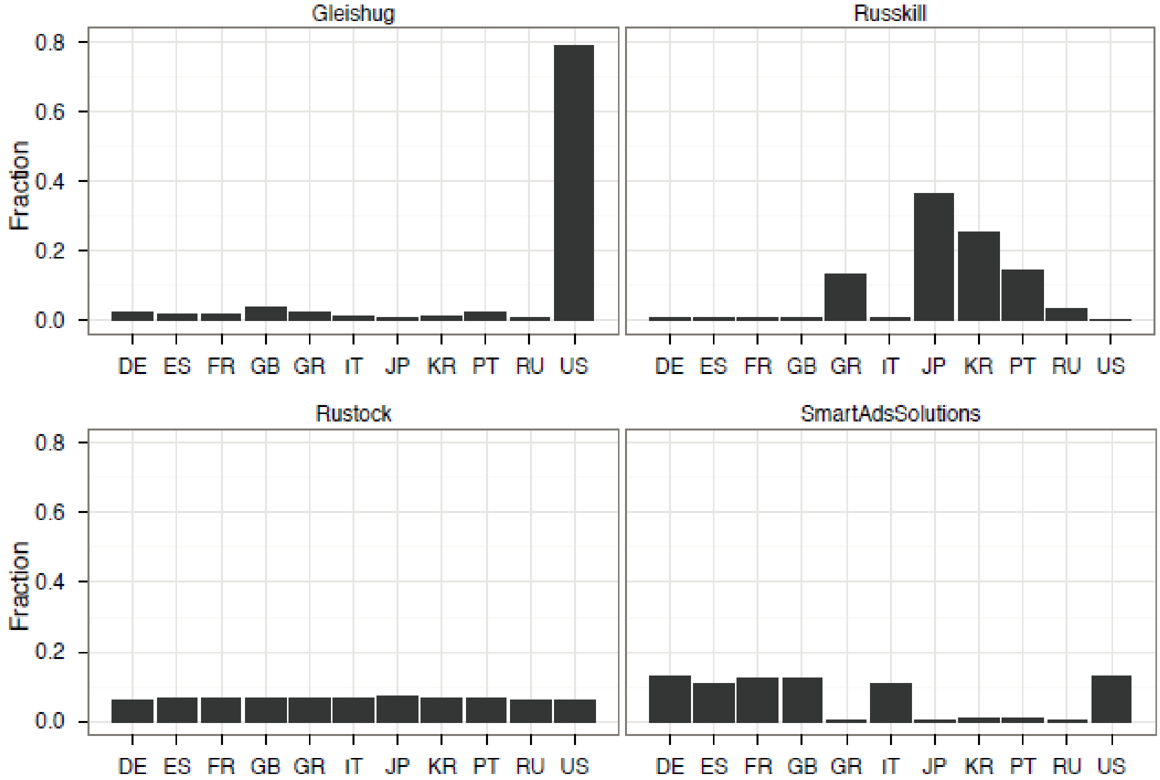
Malware Distributed per Feed

Driveby	*Dropper*	*Attachment*	*Torrent*	*Live*
Emit (12%)	Clickpotato (6%)	Lovegate (44%)	Unknown Adware.A (0.1%)	TDSS (2%)
Fake WinZip (8%)	Palevo (3%)	Mydoom (6%)	Sefnit (0.07%)	Clickpotato (1%)
ZeroAccess (5%)	NGRBot (2%)	Bagle (1%)	OpenCandy (0.07%)	NGRBot (1%)
SpyEye (4%)	Gigabid (2%)	Sality (.5%)	Unknown Adware.B (0.06%)	Toggle Adware (0.5%)
Windows Custodian (4%)	ZeroAccess (2%)	TDSS (.1%)	ZeroAccess (0.01%)	ZeroAccess (0.3%)
Karagany (4%)	Emit (1%)	Emit (.03%)	Whitesmoke (0.01%)	Gigabid (0.2%)
32 families	19 families	6 families	6 families	40 families

- Drive-by downloads compromise of choice today
  - Big Monetizers: Fake AV, click bots, information theft
- Email attachments no longer a vector
  - URLs to drive-by downloads instead
- Torrents dominated by adware

www.software.imdea.org

Geographical Distribution



Repacking Rates

- 2010:
  - 0.1 times/day (Avg.)
  - PPI dataset
- 2012:
  - 5.4 times/day (Avg.)
  - MALICIA dataset
- Sharp Rise!
- Some on the fly!

Family	Kit	Repack Rate
zbot	Kit	16.8
cridex		0.8
harebot		1.5
winwebsec	Aff	59.5
zeroaccess	Aff	18.0
CLUSTER:A		2.2
spyeye	Kit	0.6
securityshield		11.8
CLUSTER:B		30.4
CLUSTER:C		1.0
smarthdd		3.1
CLUSTER:D		3.0
CLUSTER:E		1.0
CLUSTER:F		0.7
webprotect		3.9
cleaman		7.7
CLUSTER:G		1.5
CLUSTER:H		21.7
CLUSTER:I		9.4

www.software.imdea.org

Outline

Intro

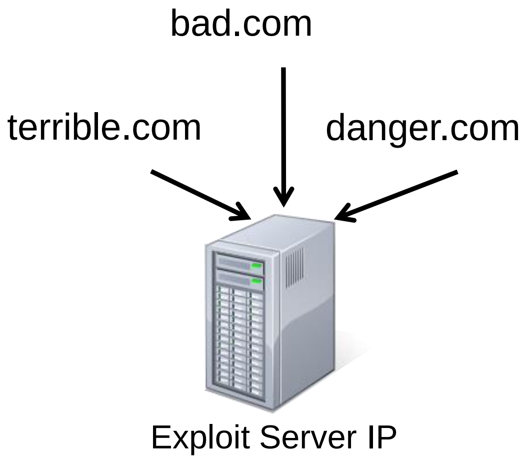
Architecture

Results

Drive-by Downloads

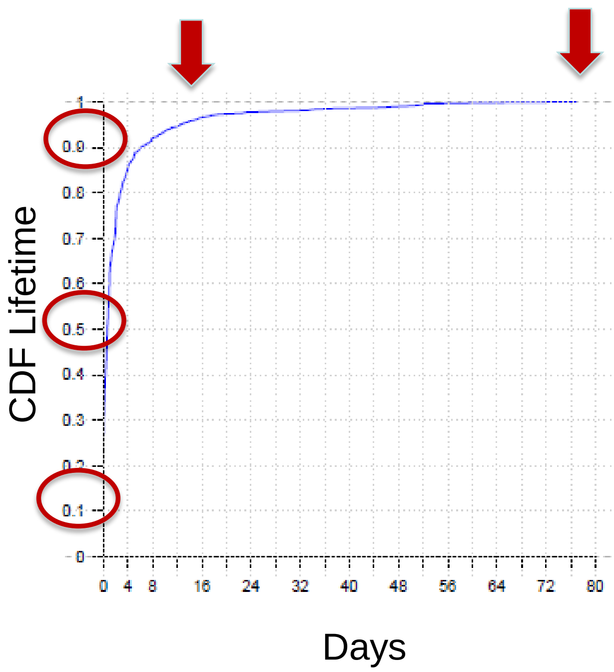
Exploit Server Lifetime

- Short-lived
  - IP :16 hours
  - Domain: 2.5 hours
- Multiple domains per IP
- Need to report both!



Exploit Server Lifetime: IP

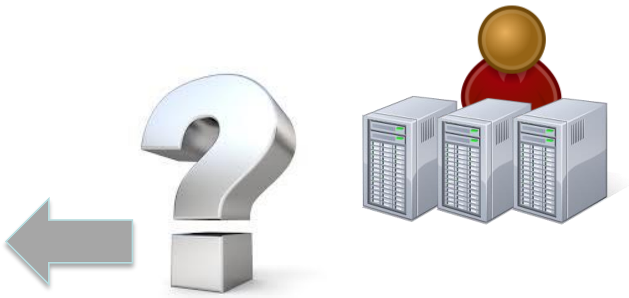
- 13% < 1 hour
- Median = 16 hours
- 10% > 1 week
- 5% > 2 week
- Max: 2.5 months



MALICIA dataset

Drive-by Downloads Operations

- 66% operations:
  - short-lived
  - 1 server



- 33% operations
  - Multiple servers
  - Servers longer lived: 5.5 days
  - Can last for weeks or months

Driving in the Cloud

- 60% of Exploit Serves in Cloud Hosting
- VPS hosting predominantly abused
- Replace dead servers with new ones

Conclusion

- Malware is a business
- Specialization in malware distribution
  - Pay-per-install
  - Exploit kits
  - Exploitation-as-a-service
- Drive-by downloads = dominant distribution vector
- Challenge and Opportunity

MALICIA Project

http://malicia-project.com

- Malware in Cybercrime
- 4 Publications
- Dataset released NEW

- Collaborators:

2.3.3 VisTracer: a visual analytics tool to investigate routing anomalies in traceroutes

EU Project Vis-Sense.

Authors Fabian Fischer, Johannes Fuchs, Pierre-Antoine Vervier, Florian Mansmann, Olivier Thonnard.

Speaker Pierre-Antoine Vervier.

Paper Summary Routing in the Internet is vulnerable to attacks due to the insecure design of the border gateway protocol (BGP). One possible exploitation of this insecure design is the hijacking of IP blocks. Such hijacked IP blocks can then be used to conduct malicious activities from seemingly legitimate IP addresses. In this study we actively trace and monitor the routes to spam sources over several consecutive days after having received a spam message from such a source. However, the real challenge is to distinguish between legitimate routing changes and those ones that are related to systematic misuse in so-called spam campaigns. To combine the strengths of human judgement and computational efficiency, we thus present a novel visual analytics tool named Vistracer in this paper. This tool represents analysis results of our anomaly detection algorithms on large traceroute data sets with the help of several scalable representations to support the analyst to explore, identify and analyze suspicious events and their relations to malicious activities. In particular, pixel-based visualization techniques, novel glyph-based summary representations and a combination of temporal glyphs in a graph representation are used to give an overview of route changes to specific destinations over time. To evaluate our tool, real-world case studies demonstrate the usage of Vistracer in practice on large-scale data sets.

BGP Event Visualizer

VisTracer: A Visual Analytics Tool to Investigate Routing Anomalies in Traceroutes

F. Fischer[1], J. Fuchs[1], Pierre-Antoine Vervier[2 3], F. Mansmann[1], O. Thonnard[3]

[1] University of Konstanz, Germany
[2] Institut Eurecom, France
[3] Symantec Research Labs, France

www.vis-sense.eu
The research leading to these results has received funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257495.

VIS-SENSE (EU-FP7)

- R&D of novel visual analytics technologies applied to network security
  - One research topic is "Visual analysis of attacks against the control plane (BGP)"

- **SpamTracer**: collection of routing data related to spam networks to study fly-by spammers

P.-A. Vervier and O. Thonnard (2013).
Spamtracer: How Stealthy Are Spammers?
In the 5th IEEE International Traffic Monitoring and Analysis Workshop (TMA), April , 2013.

- **VisTracer**: visual analytics tool to investigate routing anomalies in SpamTracer data

F. Fischer, J. Fuchs, P.-A. Vervier, F. Mansmann and O. Thonnard (2012).
VisTracer: A Tool To Investigate Routing Anomalies In Traceroutes
In the 9th Symposium on Visualization for Cyber Security (VizSec), October 2012, Boston, WA, USA.

Motivation

- CONJECTURE
  - Spammers would use **BGP hijacking** to **send spam** from the stolen IP space and remain untraceable

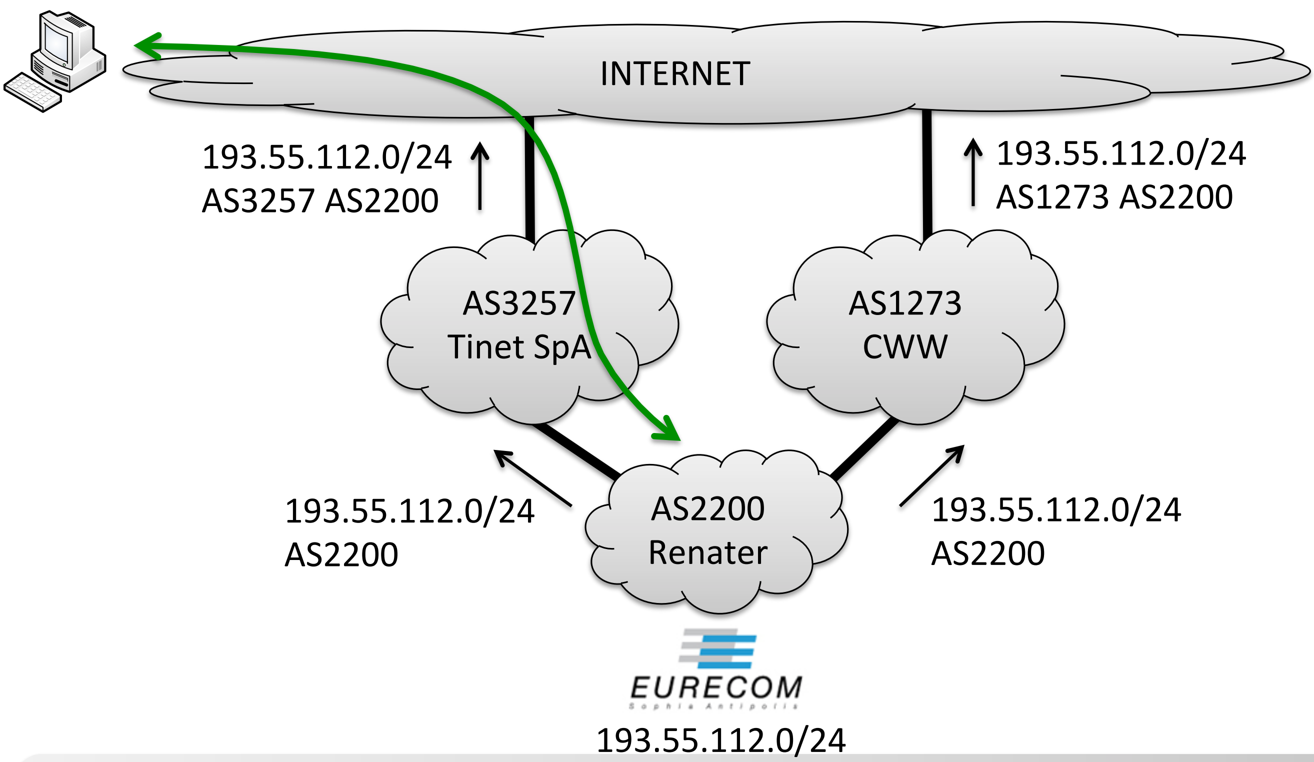
A. Ramachandran and N. Feamster (2006).
Understanding the network-level behavior of spammers.
In SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pages 291– 302, New York, NY, USA, 2006. ACM.

X. Hu and M. Z. Mao (2007).
Accurate Real-Time Identification of IP Prefix Hijacking.
In Proceedings of the 2007 IEEE Symposium on Security and Privacy, pages 3– 17, Oakland, CA, USA, 2007.

- POTENTIAL EFFECTS
  - Hijackers can steal someone else's **IP identity**
  - Spam filters heavily rely on IP reputation as a **first layer** of defense

Border Gateway Protocol (BGP)

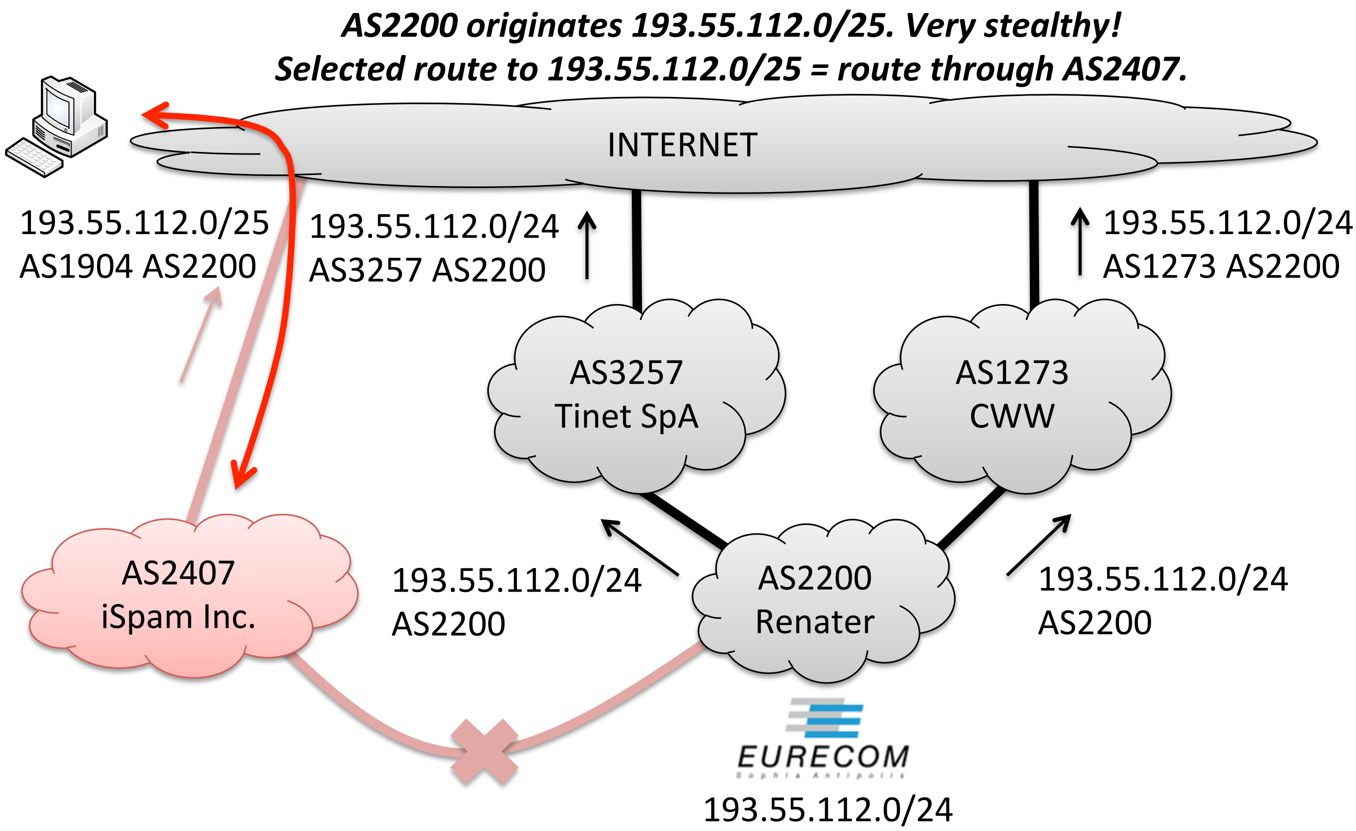
The Eurecom network 193.55.112.0/24 is originated by AS2200 (Renater, Eurecom's ISP).



BGP Hijacking ::
Or the Art of Breaking the Internet

- CAUSE
  - The injection of **erroneous** routing information into BGP
  - No widely deployed security mechanisms yet
    - Ex.: RPKI, BGPsec
- EFFECTS
  - **Blackhole** or **MITM** [Pilosof 2008] of the victim network
- EXPLANATIONS
  - Router misconfiguration, operational fault
    - Ex.: Hijack of part of Youtube network by Pakistan Telecom
  - **Malicious intent?**

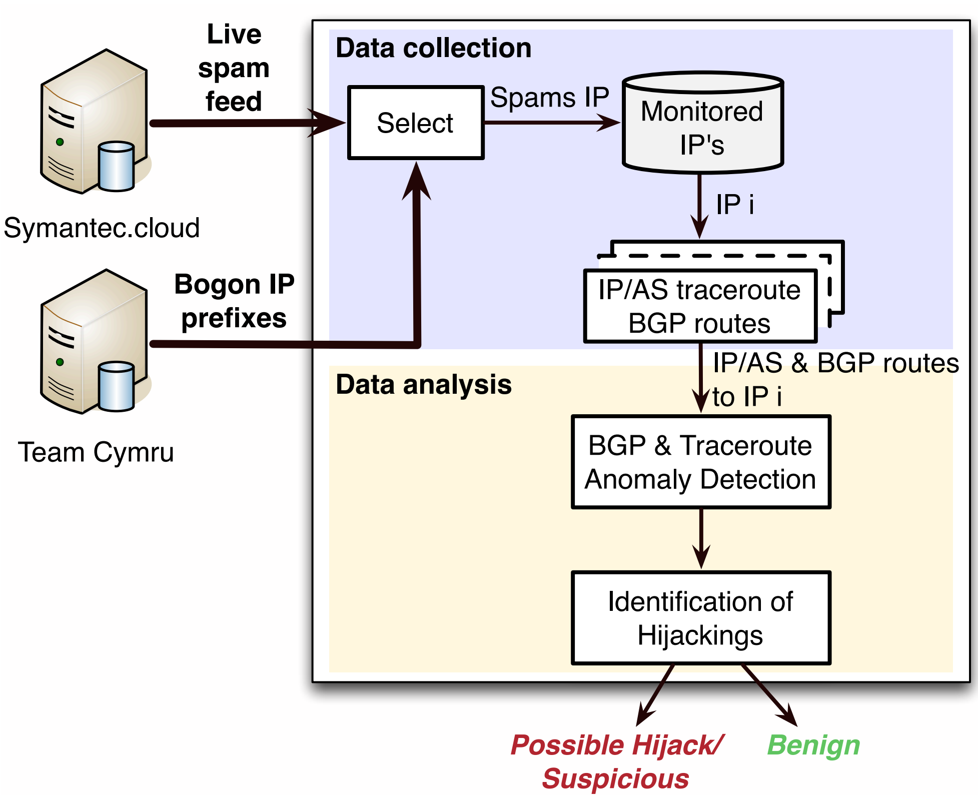
BGP Hijacking :: Example



SpamTracer :: Presentation

- ASSUMPTION
  - When an IP address block is hijacked for stealthy spamming, a routing change will be observed when the block is released by the spammer to remain stealthy
- METHOD
  - Collect BGP routes and **IP/AS traceroutes** to spamming networks just after spam is received and during several days
  - Look for a routing change from the **hijacked state** to the **normal state** of the network

SpamTracer :: System Architecture



Data Analysis

- DATA SET
  - IP/AS Traceroutes and BGP routes from SPAMTRACER
- OBJECTIVE
  - Uncover **abnormal** routing behaviors
  - Classify them as **benign/malicious**
- REMARKS
  - BGP engineering practices are similar to BGP hijacks
  - Inter-AS routing is mainly governed by private routing policies ➔ no ground-truth!

Extraction of Routing Anomalies

Prefix Ownership Conflict

Possible Reason:
Advertising someone else's IP space

Possibilities:
Same prefix (→ MOAS)
Sub-prefix (→ subMOAS)

BGP AS Path Anomaly

Possible Reason:
Changed location in Internet topology

Possibilities:
Different next hop AS
Sequence change in AS (Country) path

Traceroute Destination Anomaly

Possible Reason:
Suspicious values in traces metadata

Possibilities:
Host/AS reachability changed
Traceroute hop count changed

Traceroute Path Anomaly

Possible Reason:
Significant change in the traces path

Possibilities:
IP/AS sequence changed
Country sequence changed

VisTracer :: Graphical User Interface

Case Study 1 :: Link Telecom Hijack

The Story of a Sophisticated Spammer

- The network of the Russian ISP Link Telecom was hijacked for **5 months** (April to August 2011) by a spammer in the U.S.

- By the time their network was hijacked, Link Telecom had **suspended** their activity

- The hijacker provided the U.S. ISP Internap with a **fake proof of ownership** of the network blocks by registering the expired `linktelecom.biz` domain

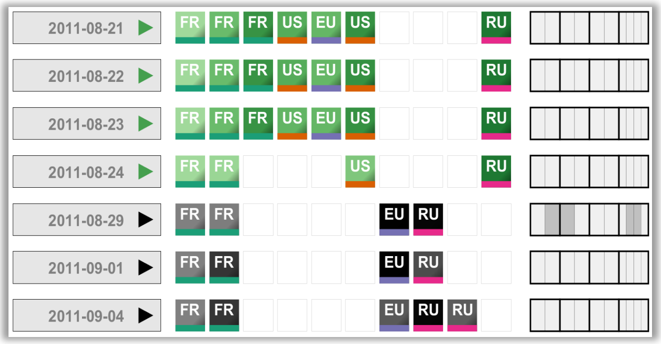
Link Telecom Hijack

Visual Exploration with VisTracer

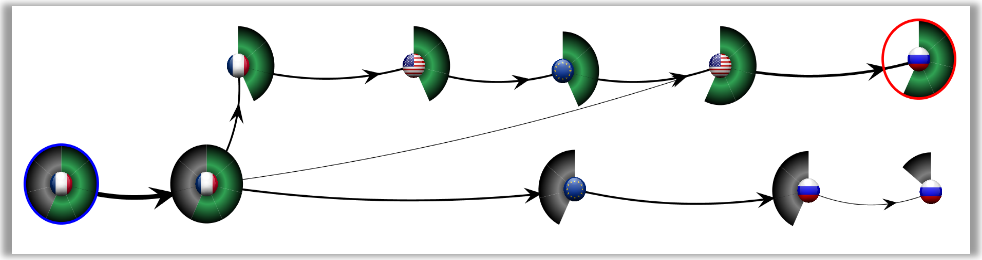
- **During the hijack:** Link Telecom's network was routed via U.S.
- **After the hijack:** Link Telecom's network was routed via Russia
- **The network administrator complained on 2011-08-20:** Observed changes were the result of the owner regaining control over his network.

More information about this case:

Symantec Internet Security Threat Report (April 2012).
Future Spam Trends: BGP Hijacking. Case Study - Beware of "Fly-by Spammers".
http://www.symantec.com/threatreport/, April 2012.



Target History Visualization shows the different traceroutes revealing the anomalies and route changes.



Graph Visualization shows the sequence of ASes traversed.

Link Telecom Hijack

Map-Based Geographic Representation

Case Study 2 :: Fly-by Spammers

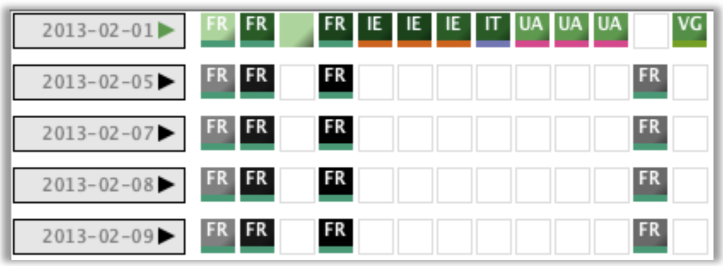
Short-Lived Hijacks By Spammers

- Link Telecom hijack was long-lived so not very stealthy because the network quickly appeared on blacklists
- Several prefixes belonging to different companies were hijacked for **1 day** to **3 weeks** for spamming
- By the time the networks were hijacked the networks had been left **idle** by their owner
- Spammers advertised hijacked networks with the **legitimate origin** AS but using a **rogue upstream** AS

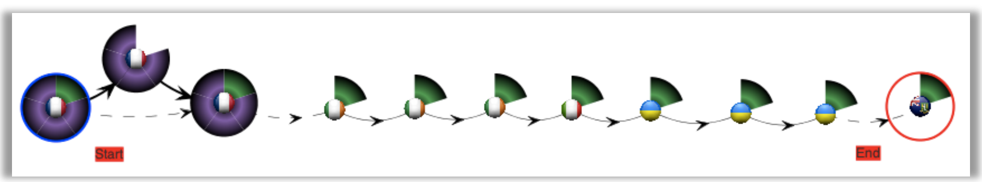
Fly-by Spammers

Visual Exploration with VisTracer

- **During the hijack:** the network was routed and responsive
- **After the hijack:** the network was **not** routed and **un**responsive
- **The network was resumed and routed for 3 weeks for spamming**
  - Observed changes correspond to the network becoming unused

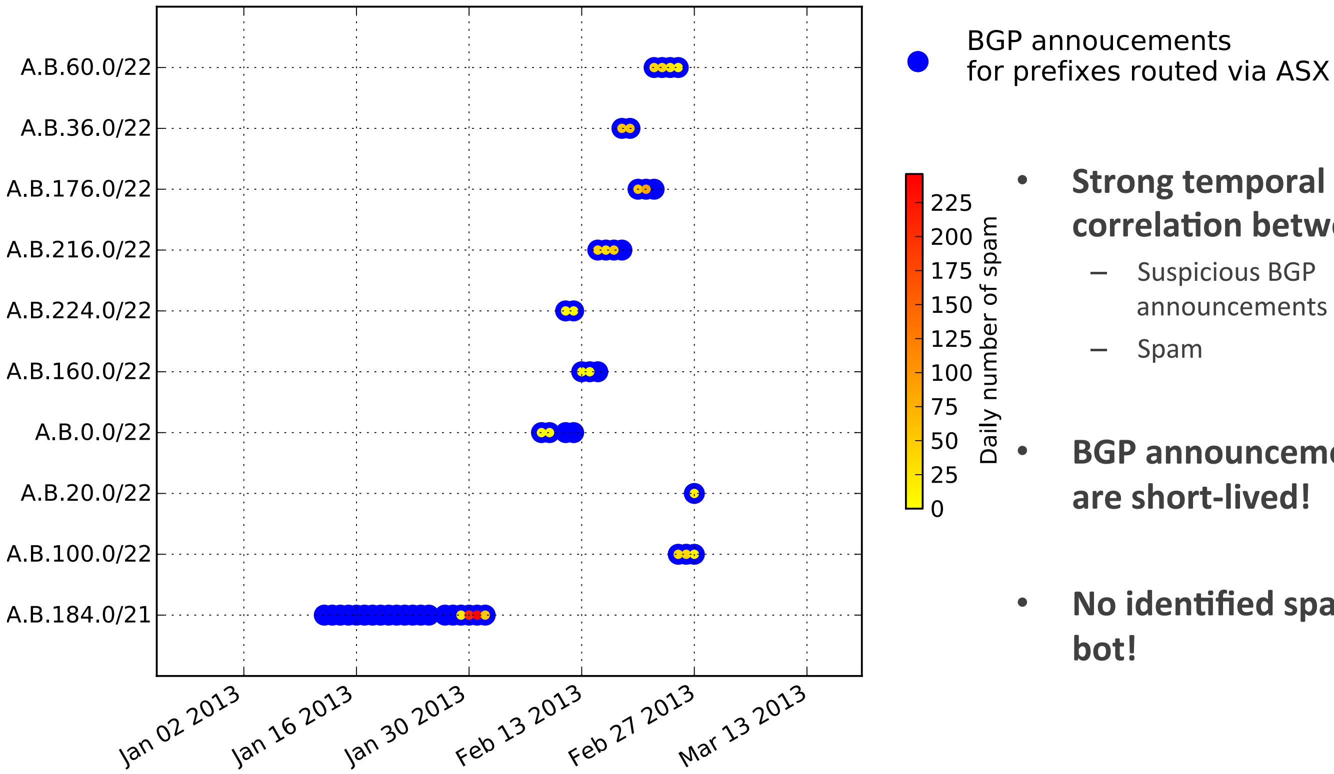


Target History Visualization shows the different traceroutes revealing the route changes.



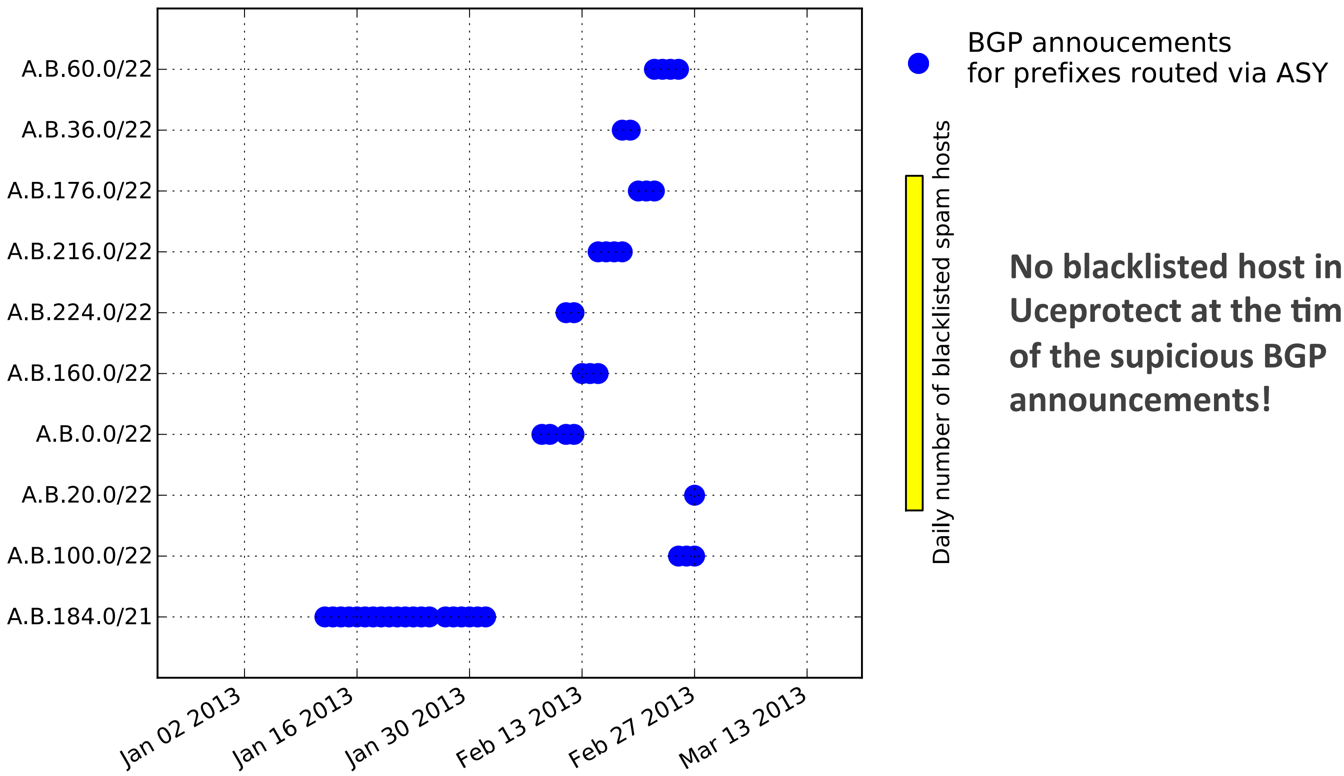
Graph Visualization shows the sequence of IP addresses traversed.

Suspicious BGP Announcements and Spam



- **Strong temporal correlation between**
  - Suspicious BGP announcements and
  - Spam
- **BGP announcements are short-lived!**
- **No identified spam bot!**

Suspicious BGP Announcements and Blacklisted Hosts

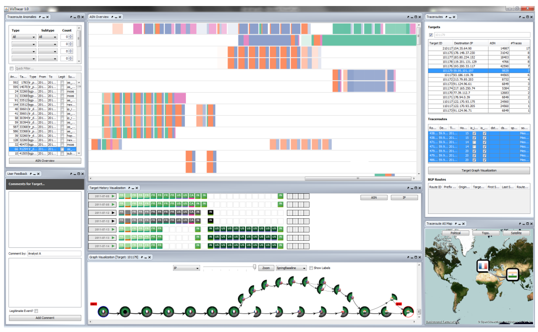


No blacklisted host in Uceprotect at the time of the supicious BGP announcements!

Conclusion

- Developed visual analytics allowed us to **uncover** and **analyze** suspicious hijack cases involving spammers
- Visualizations are **integrated** into the data collection and analysis system (SPAMTRACER)
- The several **hijackings** identified in the SPAMTRACER data set indicate behavior of fly-by spammers

Thank you very much for your attention!



Questions?

For more information
about this work please contact

Pierre-Antoine Vervier
Tel. +33 493 00 82 06
Pierre-Antoine_Vervier@symantec.com

http://www.vis-sense.eu/

VIS-SENSE
www.vis-sense.eu

The research leading to these results has received funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257495.

3 Photos Taken During the Event

To better illustrate the environment of the workshop, in this chapter we show some of the photos taken during the event, in particular in Figure 3.1 we show the final feature of the workshop, the presentation of posters by students, allowing them to receive feedback on early stages of their work by the top EU researchers present at the event.

Figure 3.1: Students talking during the poster session.

Figure 3.2: Research talks during the workshop.

4

Conclusive Remarks

In this chapter we provide list of participants and provide some conclusive remarks on this successful event.

4.1 List of Participants

In the following list, the attendees names appear in the order of registration.

- Thorsten Holz
- Felix Schuster
- Johannes Dahse
- Andreas Maa
- Nicolai Wilkop
- Markus Kasper
- Tim Gneysu
- Pawel Swierczynski
- Lukas Bernhard
- Jannik Pewny
- Hendrik Meutzner
- Juraj Somorovsk
- Tilman Bender
- Ralf Zimmermann
- Thomas Hupperich
- Andre Pawlowski
- Robert Gawlik
- Christian Rpke
- Benjamin Kollenda
- Philipp Koppe
- Behrad Garmany
- Gabor Acs-Kurucz
- Ben Stock
- Maqsood Ahmad
- Julio Fort

- Markus Schneck
- Stefan Balogh
- Fabien Duchene
- Mustafizur Rohman
- Nikolaos Karapanos
- Niko Schmidt
- Viviane Zwanger
- Vitali Regehr
- DY Yu
- richard lam
- Sree Harsha Totakura
- Jan Seebens
- Ren Freingruber
- Khaled Yakdan
- Hubert Ritzdorf
- Andreas Heydecke
- Michael Lamberty
- Mark Jeske
- Fabian Yamaguchi
- Felix Noack
- Elif Kavun
- Stephan Kleber
- Thomas Barabosch
- Patrik Lantz
- matus jokay
- Johannes Stuettgen
- Davide Maiorca

- Clemens Hlauschek
- Chris Dietrich
- christopher jmthagen
- Bjrn Johansson
- Hugo Gascon
- Arthur Gervais
- Daniel Arp
- Jens Christian Hillerup
- Christian Rossow
- Marta Piekarska
- Sb GDT
- Vida Ghanaei
- Marcos Alvares
- sergej epp
- Mahamoud SAID OMAR
- Federico Sierra
- Ulrich FAUSTHER
- Francois Crosnier
- Christian Kison
- Benedikt Driessen
- Sebastian Lekies
- Paul Irolla
- Ugur Cihan KOC
- veysel hatas
- Thomas Petig
- Ivan Pustogarov
- Pierre WILKE

- Christian Kudera
- Federico Maggi
- Anastasia Skovoroda
- Bruno Berger
- Charles Lim
- Zaky Nurahman
- Eros Lever
- Andrea Scorti

4.2 Conclusions

The workshop was well received by the participants, who attended both the talks and the poster session with interest, engaging in brainstorming and networking activities among them as well as with the speakers and teachers.

Thanks to this second workshop we showed to the system security community the results of the SysSec activity: Several outstanding papers involving SysSec partners or associate members were published in the proceedings of top venues, showing the excellence of the people involved directly and indirectly in the consortium.

Co-locating the workshop strategically at the UbiCrypt Summer School allowed us to reach the young minds that will be part of the future of our system security community, hopefully continuing our work.