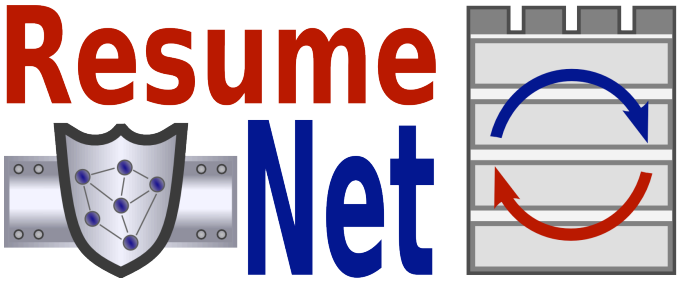
Resilience and Survivability for future networking: framework, mechanisms, and experimental evaluation



Deliverable number	1.6
Deliverable name	Collaborative crosslayer monitoring as resilience enabler
WP number	1
Delivery date	31/12/2011
Date of Preparation	11/1/2012
Editor	Andreas Fischer
Contributor(s)	Andreas Fischer, Michael Till Beck, Hermann de Meer (UNI PASSAU)
Internal reviewer	Sylvain Martin, Marcus Schöller

Summary

Distributed correlated network monitoring is important for network resilience in order to identify faults and threats that are otherwise hard to track. In particular, challenges involving a combination of different services may pose a problem for troubleshooting. In order to diagnose the root cause of a given challenge, multitudes of monitoring tools/methods are in use, each having its own advantages and disadvantages. Nowadays, end-user services are often combined from a multitude of network services and will fail if one of those provider services fails. From the user perspective, it is difficult to remediate these challenges without intricate knowledge about the underlying process. Cooperative correlation of events on different network layers is an important building block in uncovering complex challenges. This deliverable presents a use-case scenario and highlights the importance of distributed correlated network monitoring that enables information to be gathered from multiple layers and renders networks and services resilient to challenges, such as attacks and failures.

Contents

1 Introduction

With increasing complexity both in infrastructure as well as in services it is vital to also increase the level of protection against intentional and unintentional disruptions. Due to the increased system complexity it is, however, quite difficult to diagnose the root cause of a given problem. Thus, the complex and wide-spread nature of today's networks demand the capability of remotely managing infrastructural elements and technical service providers such as servers and/or network accessible storage devices. Though, while being secured by standard security mechanisms such as passwords and cryptographic measures, these remote management interfaces into the aforementioned systems are quite often exposed to attacks from outside and there is a likelihood of them being exploited. Hence, these vulnerabilities in the network can be exploited for purposes such as cyber-terrorism, industrial espionage, identity theft and other malicious uses. Therefore, when relying on infrastructural elements to support the transmission of information over such complex, but vulnerable networks it is imperative to not naively trust the integrity of all involved network nodes.

Traditionally, a lot of information was collected locally, by inspecting incoming network packets and monitoring application log files. However, there is only limited information available this way. Network packets may be faked and applications may miss the important details in their log files. Having several hosts cooperate in getting information about network traffic will lead to a far more accurate view of network activities. In particular being able to trace the datapath of a certain action by having each host involved generate some information about the data it handles, may reveal information to any interested party about:

- The existence of unexpected traffic, realizing that data took a path in the network it was not expected to take
- The source of malicious traffic, allowing to trace back a networked attack to its initiator or at least giving an indication of who is a possible suspect and who is not.
- The failure of individual nodes to provide the expected answer/result to an incoming message.

Network monitoring mechanisms are used to collect and analyze network traffic data for network management. At present a number of network monitoring/diagnostic tools are available but many of them are limited to specific protocols. Different network monitoring tools have been developed and used by different entities depending upon their specific needs. For example, there are monitoring tools that collect time series system data from a set of nodes and allow operators to identify hotspots, and high-level bottlenecks such as disks, CPUs, or network failures. Similarly, there are open source monitoring tools which are used to collect "health data" from hosts, services, and network components. In summary, monitoring should give an insight into a running system and provide system developers and administrators with a facility to spot and resolve the failures. This is an important aspect for any network monitoring tool in today's complex and distributed network environment. Especially for large distributed systems where there is a possibility that multiple components may fail independently or it may happen that communication latencies may violate the design-time assumptions. In such cases it is important to record and examine faulty executions and also to understand the cause of unexpected behaviour.

This deliverable builds on and extends research we already reported in ResumeNet Deliverable D1.4 [BBF+10]. It takes into account the work on monitoring presented in ResumeNet

Deliverables D2.2b and D3.2 [SFM+10, DDK+10]. The work presented in this deliverable extends previous work by stressing the need for monitoring mechanisms that are both cross-layer and distributed and are able to correlate events.

2 Monitoring as part of a Detection framework

2.1 Distributed Network Monitoring

A Distributed System can be considered as a type of computer network that spans across multiple administrative boundaries. Using distributed systems, various computer programs can be run or executed remotely irrespective of physical locations. This enables the utilisation of memory and processing power of remote computers. Though, on the one hand distributed networking increases speed, memory and processing power, while on the other it becomes more vulnerable to attacks. Hence, they require much more efficient ways of monitoring in order to maintain constant service availability. However, it is quite difficult to understand how any given client request is being fulfilled within a service and why other requests fails. For the reasons mentioned above we need distributed network monitoring. Distributed network monitoring involves multiple "pollers" distributed around the network, reporting on events from multiple locations in the network. With single point network monitoring one can see the network from a single perspective. With distributed network monitoring one can see the network from a number of different views across the network, i.e, it provides a global view of the network. Distributed network monitoring therefore allows for a deeper understanding of the network, giving us the ability to detect outages and bottlenecks more easily.

Internet applications are becoming increasingly distributed and complex and so the challenge of developing, deploying, managing and troubleshooting them is also increasing. In order to deal with all the aforementioned problems and to make the current network more robust and resilient against challenges, distributed correlated network monitoring mechanisms are required.

Distributed correlated monitoring involves multiple points (cross-layer, cross-domain) distributed across the network for measuring network performance at multiple locations in the network. Distributed correlated network monitoring is important because it helps in the collection of useful information across different network layers and administrative domains of the network so that the network can be managed and controlled using the collected information. Such network monitoring techniques are required to allow network monitoring applications to check the states of their network devices. In order to achieve the tasks mentioned above, it is important to make use of monitoring tools which are task-centric rather than device-centric. Unlike device-centric monitoring tools, task-centric tools enable an operator to causally trace the complete execution of a networked system across the boundaries of applications, protocols, and administrative domains. Furthermore, as more and more network devices are used to build bigger/complex networks, network monitoring techniques need to be expanded for monitoring the network as a whole. Distributed correlated network monitoring is required because:

- Network devices are becoming more and more complex
- The network infrastructure is becoming more and more complex with the interrelated services

In the light of the above statements, we argue that causal, end-to-end task monitoring must

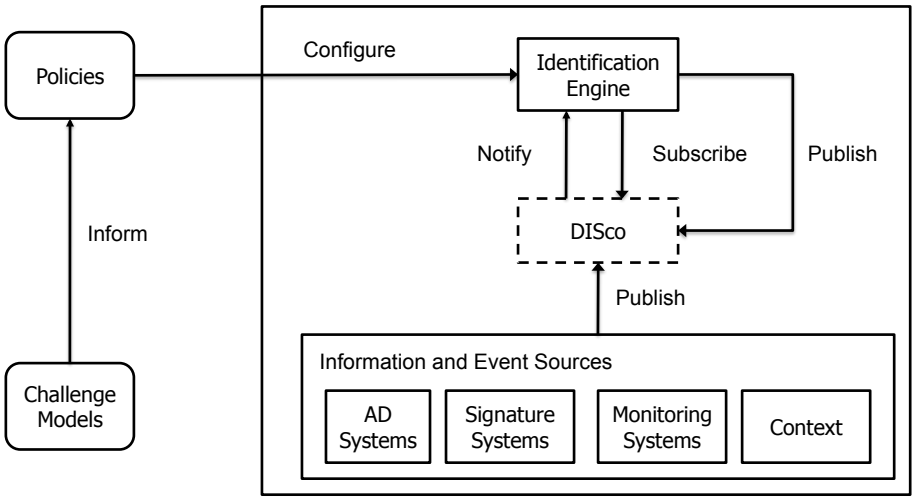


Figure 1: ResumeNet Challenge Identification Architecture

be an integral part of network services. On the one hand with such a monitoring mechanism it is possible to:

- Follow a request from start to end, with variable level of details across applications and network layers
- Log the relevant information from the devices connected with each tagged operation, which can then be reported back and provide a comprehensive view of what network operations have been executed as part of the task.

On the other hand task-centric monitoring tools:

- Can provide causal, end-to-end visibility into network services and hence enabled the discovery of a number of bugs and helps in diagnosing the performance faults
- Can be incrementally deployed and should be compatible by components provided by different parties, much like services themselves

2.2 Network Monitoring in ResumeNet

The $D^2R^2 + DR$ strategy emphasizes the importance of challenge detection by making it an explicit element of the inner control loop. Detection extends purely passive defense mechanisms by striving to identify challenges as early as possible in order to trigger a subsequent remediation. Challenge detection relies on the monitoring and analysis data flowing throughout the network. Challenges to network operation can have an impact in various locations in the network. Therefore it is important to have monitoring facilities placed at strategic locations along the network path that allow to collect data both cross-layer and cross-domain.

Figure 1 shows the basic building blocks of Challenge Detection in the ResumeNet model. Monitoring data is collected throughout the network and distributed to a distributed information storage facility, represented by DISco in Figure 1.

The so called Identification Engine then subscribes to the information provided by DISco and performs the necessary analysis on the data. This analysis will be parametrized based on

Challenge Models and Policies from outside the actual Challenge Identification Architecture. The results of the analysis are then fed back into the information storage, providing input for remediation actions.

Monitoring plays an important role in this scenario. It is necessary to get a comprehensive view of a system in order to prevent a faulty diagnosis by the Identification Engine. This becomes a challenging task when dealing with large, distributed systems composed of several networked components. Distributed network monitoring mechanisms therefore are an important building block for challenge detection in such environments.

3 An overview of network monitoring mechanisms

A variety of network monitoring tools focuses on network monitoring. We have already given an extensive list in ResumeNet Deliverable D1.4 [BBF+10]. There are five more pieces of related work, which have not been mentioned in D1.4, yet, and therefore shall be mentioned here.

Braun et al. [BDKC10] summarize current packet capturing solutions. The authors identify bottlenecks within the capturing stack of FreeBSD and Linux and provide guidelines on how to configure these systems for optimal performance. Castro et al. [VC11] present an overlay that is used for network monitoring. Measurement and redundancy is reduced by the composition of nun-overlapping measurement paths. This ensures that the approach remains scalable. Errais et al. [EBRR11] discuss a distributed Architecture for IP Multimedia Subsystem (IMS) monitoring. Their approach tries to provide permanent surveillance and supervision of services. Habib et al. [HKB04] try to detect congestioned links inside a network. They aim to detect bandwidth theft attacks and service violations. Their approach tries to minimize the number of probes that are needed to detect attacks.

Fusco et al. describe the design and implementation of a scalable multi-core aware packet capture kernel module [FD10]. They present common pitfalls of network monitoring applications used with multi-core systems. A Packet capturing technology tailored for modern multi-queue adapters is discussed. This simplifies development of highly scalable multi-threaded traffic analysis applications and is expected to minimize the memory bandwidth footprint.

4 Distributed monitoring constraints

4.1 Assumptions for reliable detection

4.1.1 Horizontal metadata coverage

In order to reliably trace a task, at least the source and sink nodes should support the generation and processing of monitoring metadata. This is a minimum requirement. In order to support high granularity during the monitoring process, all nodes participating in the processing of a task should have the ability to read and generate monitoring metadata.

4.1.2 Cross-layer metadata coverage

To successfully identify faults in the network, it is not enough if only the application layer of the network stack generates monitoring metadata. Failures caused by lower layers on a node

somewhere between the end nodes would still be hidden from a communicating node, making it impossible to determine the cause of the failed message transmission. Only if all network layers participate in the processing, generation and propagation of monitoring metadata can a conclusive description of the network flow for the message in question be generated.

4.1.3 Reliable access to reports

To enable the detection mechanisms to generate sufficiently reliable results, monitoring report data that is sent out-of-band from normal message flow has to be sent using a reliable protocol. If this prerequisite is not given, the analysis stages would not be able to detect if a report was lost in transmission or not generated or sent. If the nodes use a reliable protocol to distribute the reports to the requesting node, a report that fails to transmit indicates a very unreliable network connection to the node in question which would make it beneficial for the network as a whole to exclude the node from routing. So given a reliable transmission protocol, missing reports from individual layers or a node as a whole can be said to indicate malfunction or malice and thus increase the node's distrust score.

4.1.4 Performance constraints

Each node participating in the generation of monitoring metadata has to spend some of his resources for the task of generating reports. On the one hand, processing cycles will be spent for the analysis of network traffic. On the other hand, monitoring data has to be stored for later retrieval. This imposes a certain overhead on the monitoring node, which has to be taken into account.

4.2 Means of problem identification

As the generation of monitoring metadata for network messages implies CPU and bandwidth overhead, it may be advantageous to not monitor all activities. There are several schemes to choose when to monitor. Two of them will shortly be discussed here.

Probe-On-Suscpicion Analysis In a probe-on-suspicion scheme, one of the participating communication partners initiates monitoring when the initial action did not yield the expected answer. This way, the node applies an optimistic detection scheme, expecting all hosts to be functioning according to the defined protocol, until it actually reveals a malfunctioning behaviour towards the network. While this will, in a network with a sufficiently small number of malfunctioning nodes, decrease the traffic and payload overhead dramatically, it fails for faults which will occur only sporadically.

Random Pick Analysis In a random pick analysis scheme, a host randomly probes the message paths of it's transmissions for network malfunction. Using several parameters to prevent over- and under-investigation of a given route, the node would again need to rely on local memory being available to the client to store statistical data on monitoring. Using this scheme, the node can pro-actively scan the network for malfunctioning nodes, identifying them directly when they fail. Combined with the Probe-On-Suspicion scheme, this creates a high probability of error detection without generating too much excess network payload.

4.3 Monitoring with malicious users

Malicious users in a network can have a severly detrimental effect to monitoring reliability. It is therefore necessary to discuss the possible influence of attackers on a given monitoring system. Here, X-Trace will be used as an exemplary distributed monitoring mechanism.

Considering the possibilities of a potential attacker, several constellations are possible. First, different attackers will have different levels of control over a computer system. This may or may not hamper their possibilities with regard to suppressing logging information. Second, attackers can be differentiated by looking at the number of computer systems under their control. Having more than one system available opens up several additional options for the attacker.

Regarding control over the computer system A (human) attacker might not have full control of the computer system he uses. With regard to the TCP/IP stack, one can classify attackers, according to the level of control over the network stack they have on their machine. In most modern operating systems control over the transport layer and lower layers requires elevated system privileges. Thus, we can differentiate between two kinds of attackers:

1. Attackers controlling the application layer but lacking control of transport and network layers (regular users)
2. Attackers having full control over their respective system (system users)

Attackers that are regular users can suppress information generated by the application layer - in particular regarding network traffic initiated by themselves. However, they cannot hinder their network layer generating reports on incoming packets carrying X-Trace metadata.

On the other hand, attackers that are system users are able to suppress both, information generated by the application layer as well as information generated by lower layers (including, e.g., information about forwarded packets generated by the network layer).

Regarding number of controlled computer systems An attacker might not actually be restricted to using only one computer system. Again, there are several possibilities:

1. Attackers controlling only one computer system
2. Attackers controlling an entire AD
3. Attackers controlling several distributed computer systems
4. Attackers controlling several distributed ADs

In the first case an attacker may be able to suppress X-Trace information originating from his host, but the AD this computer belongs to is still able to provide extensive information on packets passing it, without having the attacker modifying this information.

In the second case the attacker is able to alter recorded information within his AD. While other ADs may provide enough information to identify the malicious AD, identifying the attacker still becomes harder.

Controlled Systems	X-Trace initiated by	Attacker is regular user	Attacker is system user
Single system	Attacker	Caus. rel. broken (App. layer)	Caus. rel. broken (host)
	Defender	Attacker identified	Host identified
Admin. domain (AD)	Attacker	App. layer info misleading	AD identified
	Defender	App. layer info misleading	AD identified
Distributed systems	Attacker	Caus. rel. broken (App. layer)	Caus. rel. broken (host)
	Defender	Attacker identified (1 host)	Host identified (1 host)
Distributed ADs	Attacker	App. layer info misleading	AD identified (1st only)
	Defender	App. layer info misleading	AD identified (1st only)

Table 1: Implications of different kinds of attackers on X-Trace

In the third case an attacker may be able to obfuscate routing by removing all X-Trace metadata embedded in incoming packets at the first node under his control. Traffic will however still be traceable to this first node.

In the last case only the AD nearest to the target which is under the control of the attacker can be identified. All the attacker has to do is to stop all X-Trace metadata processing on incoming packets and, if X-Trace is required for outgoing packets, start logging at a possibly unrelated node within that AD.

Implications on X-Trace All in all, the situation depicts itself as summarized in table 1. If the attacker controls only a single system and does not have system privileges, the causal relationship is broken on the application layer when he is the one initiating the X-Trace process. If the defender initiated the X-Trace process, the attacker is identified without any problems.

If the attacker has system privileges and he was the one initiating the X-Trace process, the causal relationship is broken even down to the network layer. If the defender initiated the process the attacking host might be identified, depending on the trustworthiness of the rest of the administrative domain the attacker is part of.

If the attacker controls an entire administrative domain and he is a regular user on each computer, X-Trace information will be correct on the network layer but may be misleading on the application layer. This situation will be quite unlikely, though. On the other hand, if the attacker has system privileges, the host where the attack originated is hidden, although the administrative domain may be identified, depending on the trustworthiness of the administrative domain adjacent to the attackers administrative domain.

In the case of an attacker controlling several distributed systems (like in the case of a bot net, for example) and initiating the X-Trace process, he can reroute his packets through several systems under his control, using them as proxy servers. This will however still leave the last node open for investigation with about the same properties as with an attacker restricted to a single system.

If the defender was the one initiating the process, the situation also is more or less the same as with an attacker controlling only a single system - with the restriction that traffic will only be traceable to the first system under the attackers control. The fact that the attacker

controls more than one systems remains hidden for the investigator. Anyway, getting info about the first host under the attackers control may already be a valuable first step to uncover his schemes.

Finally, an attacker controlling several distributed administrative domains will easily be uncovered on the network layer. If application layer routing is involved, at least the first node / administrative domain under his control will be uncovered. If he has system privileges, the situation is similar: whether he or the defender is the one initiating the X-Trace process - the attacker will only be traceable to the first administrative domain under his control.

4.4 Security assumptions

While X-Trace can be used to investigate different kinds of attacks, there still remain some problems and implications that have to be taken care of.

Trusting other administrative domains Information generated by X-Trace, is recorded separately for each administrative domain (AD). While necessary in order to take each ADs needs for security and privacy into account, it opens up a problem: Which ADs should an investigator trust? By controlling the recorded data, an AD cooperating with the attacker has the theoretical possibility to alter the data in order to counter investigation. A trust infrastructure is needed to allow investigators to decide, which information to use and which information to discard. While the details of such an infrastructure are not within the scope of this document, in the end, there will be at least trusted ADs and untrusted ADs. Information gained via X-Trace is then only reliable, as long as it comes from trusted ADs.

Let us regard a situation, with several trusted as well as several untrusted ADs. In that case, as long as the traffic traverses only trusted ADs, everything is fine. But as soon as untrusted ADs are affected, the task tree starts to corrupt as information by untrusted ADs has to be discarded as unreliable. The more untrusted ADs are out there, the more incomplete is the reconstructed path. In that respect, untrusted ADs act just like ADs under the control of an attacker: any information they provide can not be used.

In order to assess the status of any AD, a trust relationship has to be maintained. Such a relationship might either be a static one, defining once (possibly manually) which ADs are trusted and which are not, or it might be dynamic, altering the status of an AD according to certain metrics.

As an example for such a metric consider an untrusted AD "U", that is surrounded by trusted ADs. Any traffic into and out of U will be reported upon by the trusted ADs. If the information that is provided by U is then in line with what the trusted ADs reported, U might gain some credibility, since it seems that - at least concerning the edges of the AD - U didn't lie this time. Over time, an AD that always seems to speak the truth, might eventually get "trusted" status.

Of course this is subject to the usual pitfalls inherent in trust systems: U might pose as an honest AD for a very long time, only to deceive others in believing it once it gets malicious [MGM06]. This might even not be the original intention of the AD (or rather its owner) - consider for example an AD that has been hacked secretly.

In any case, a certain level of knowledge about the underlying network infrastructure may be required in order to determine which ADs to ask for information. While in an ideal case, where every AD is trustworthy, it will suffice to trace the path backwards to its source, asking

each AD consecutively for information, this does not hold once an AD is regarded to be untrustworthy: potentially false information about whether the traffic originated in that AD or not has to be checked with adjacent (hopefully trustworthy) ADs, requiring knowledge about which ADs are adjacent to the AD in question.

Completeness of the task tree Unlike when using X-Trace for error detection, using X-Trace for detection of malicious behavior necessarily always results in a more or less corrupted task tree. All data coming from entirely untrusted domains that were crossed during transit, is itself not reliable and should be deleted from the tree. Regarding the attacker himself, the information provided by him is certainly doubtful, if not outright false.

Depending on the capabilities of the attacker, several parts of the task tree may already be gone. Additionally, any ADs that are not trusted by the investigator (but are not necessarily involved in the current attack) are supposed to not provide reliable data (or else they would be trusted). This leads to even further corruption of the task tree.

Still, although causal relationships may become harder and harder to see, the possibility remains to take the corrupted task tree and place all nodes on a timeline, thus getting at least a notion of the order in which events happened.

Apart from that, it may well suffice to be sure whether a packet did or did not cross a certain network in order to either identify the attacker or at least rule out a few innocent bystanders.

Level of X-Trace deployment Obtaining useful information from X-Trace requires a certain level of X-Trace deployment as well as cooperation between different administrative domains. The less deployed X-Trace is in a certain scenario, the more corrupted the resulting task-tree will be. The problem is actually similar to the trust problem discussed in Section 4.4. Each administrative domain, that either denies access to its X-Trace database or does not support X-Trace at all, is equivalent to an AD, that is entirely untrustworthy (in which case the data is there, but invalid).

Still, X-Trace does not require complete deployment throughout the entire network - if there is a sufficient level of deployment within a certain context (like within a corporate network), X-Trace can already help investigating problems in that context.

5 Scenarios

This section illustrates three scenarios, where collaborative distributed network monitoring is necessary to detect a fault in the network. X-Trace is used as exemplary mechanism for network monitoring here.

5.1 Hidden DNS failures

Distributed correlated network monitoring is important for network resilience in order to identify faults and threats that are otherwise hard to track. In particular, challenges involving a combination of different services may pose a problem for troubleshooting. Nowadays, end-user services are often combined from a multitude of network services and will fail if one of those service providers fails. From the user perspective, it is difficult to remediate these

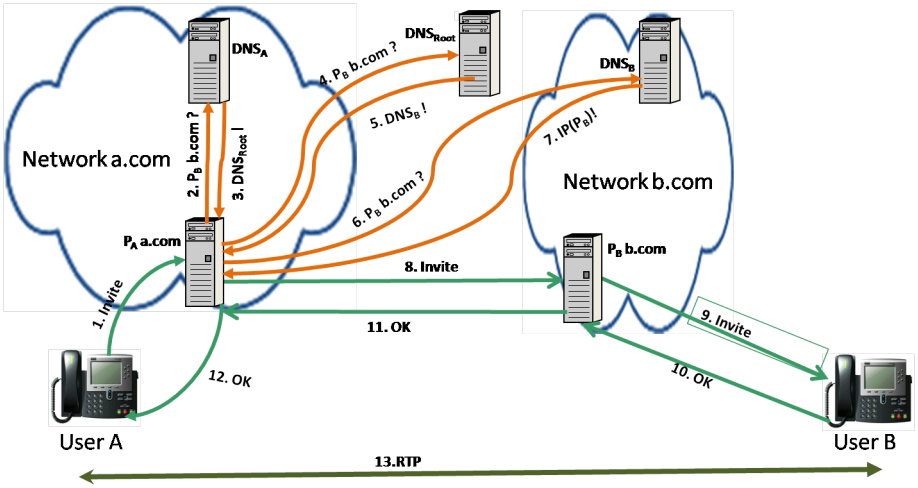


Figure 2: A SIP scenario

challenges without intricate knowledge about the underlying process. Even in the simplest example, where a user contacts a service S_1, which in turn depends on service S_2, it is hard for the user to identify a fault occurring between S_1 and S_2. These problems may be further complicated by the fact that communication challenges may be limited to a particular situation or scenario, as can be demonstrated by the following scenario.

We will assume two users, User A and User B. User A is trying to set up a VoIP session with User B using the Session Initiation Protocol (SIP). This involves several steps, as shown in Figure 2. First, User A sends a SIP Invite message to his local SIP Proxy P_A, asking the Proxy to locate and contact User B. P_A will then first try to identify the responsible proxy for User B (P_B). This may require a DNS lookup, possibly involving iterative queries, in order to get the IP address of P_B. Once the IP address is available, P_A will forward the SIP Invite message to P_B who then subsequently forwards it to User B. If User B is willing to accept the call, a SIP OK is sent to User A via P_B and P_A and, finally, the call is established.

Now, one can assume that the DNS message from DNS_B to P_A does not arrive. Note that here an iterative DNS approach (as opposed to a recursive one) is assumed. The loss of a message in this scenario can be due to several reasons. It could simply be due to an overloaded link on the path, causing packet drops. In this case, the problem can typically be remediated by trying again after some timeout. However, the problem can also be due to a persistent underlying challenge. This will cause failure not only of one try, but also of all subsequent tries. This could be due to, for example:

- A misconfigured packet filter causing packets matching a certain profile to be dropped (e.g., all outgoing UDP packets).
- Misconfigured routing tables causing connectivity problems between DNS_B and P_A. Note that, in this case, connectivity between other parties might still work without problems.
- A bug in the DNS software of DNS_B, causing the software either to crash or not to produce any response. Note that, in the latter case, the DNS server still might be able to answer other requests correctly.

For all these problems, the final result would be a failure in setting up the call from User

A to User B. From the point of view of User A, such a problem is very hard to identify, as the root cause of the problem is an indirect result of his request. Moreover, the actual problem (missing connectivity between P_A and DNS_B) does not actually involve User A. In order to correctly diagnose this problem, events throughout the network have to be correlated both across different networks and across different network layers (i.e. both vertically and horizontally).

Note that, while the given example is very specific, the underlying concept stays the same for different scenarios. In general terms, the problem can be stated as follows:

1. Two parties want to communicate (here: User A and User B)
2. This request triggers an (indirect) action between two other parties (here: P_A and DNS_B)
3. The latter action fails due to a problem in the network
4. Subsequently, the communication request fails.

By looking at this scenario, the need for distributed correlated network monitoring becomes clear. Isolated, non-cooperative monitoring mechanisms are unable to properly detect this kind of fault and communicate it back to the user.

5.2 Identifying malicious routing behaviour in overlay networks

As overlay networks define a virtual network upon a physical one, routing is a central element both in protocol design for networks as in laying out attack schemes against said networks. Overlay networks introduce a new virtual layer on top of the traditional TCP/IP stack positioned within the application layer of the stack. Within this virtual layer, an additional layer of routing is introduced to ensure eased communication between members of the overlay network regardless of the underlying transport technology. As such, the number of possible points of failure and possible targets for attacks increases dramatically compared to a standard network as an attacker can target both the traditional stack as well as the virtual one.

A classical problem in traditional routing is the malicious dropping of network messages passing through a network router commonly resulting in network disruption, increased network traffic and decrease in message throughput. The same can be said for overlay routing mechanisms in general though some implementations may be more susceptible to this matter than others. Following the concept of overlay networks, the problem of routing misbehaviour in overlay networks is twofold. The routing may be tampered with in the underlay network, in the domain of IPv4/IPv6 routers, firewalls and local network stack implementations. On top of that, the forwarding process might be manipulated within the overlay layer itself, resulting from a modified peer implementation of the protocol. The details of the misbehaviour, where and why the packet was lost, are, by design, invisible to the overlay network. All a node participating in message forwarding would be able to recognize is that a message failed to reach its destination. It would be impossible to tell whether this was due to a faulty landline, a malicious underlay router or an overlay node refusing to forward the packet within the overlay protocol. The underlay part of this problem has already been addressed in numerous articles and papers with a multitude of possible solutions and detection mechanisms. Here it is discussed how to detect routing misbehavior in overlay networks.

The idea behind the concept proposed is to enrich P2P messages, or, more general, overlay messages with X-Trace metatada and collect reports from nodes participating in message propagation using predefined instrumentation strategies, for example probing on suspicion (as described in 4.2). The returned reports are collected in an in-memory database for offline reconstruction of the message path and following analysis. By comparing the reports received, the investigating node can easily determine if the message reached it's target as expected and, if that is not the case, where on the path the message appears to have been lost. While it would be easy for a malicious node to fake a report that the message arrived at it's destination and trick the investigating node into believing that nothing happened, the reports are generated on multiple layers and the discrepancy between the network address of the node and it's hash address would be detecable in the offline analysis, unveiling the misbehaviour.

The malicious discarding of network messages within overlay networks is a simple means for an attacker to disrupt network operation and negatively influence the throughput of the whole network. All the attacker has to do is to refuse to participate in the normal packet routing protocol defined by the overlay network he is a part of. If unhandled, malicious message drops can severely hinder the correct functioning of nodes within the network or even bring the network's throughput to a complete halt. So the question is raised how such a malicious or ill-functioning node can be identified and, in effect, be blocked or isolated, messages being re-routed around the node. Similar questions have been raised for quite some time in the field of normal "underlay" network routing environments which, in answering, have found a number of solutions.

A node in a P2P overlay network that refuses to route messages to other participants is easily identified. The misbehavior is obviously detected by packets that were not received correctly. X-Trace can then analyse the path the packets took and point to the last node which received the missing packets. The same holds true for a node that reroutes messages to nodes not intended to receive the message. If the message does not turn up at the receiver, an X-Trace analysis will easily point to the node who misplaced the message. The difficulty in that situation lies in the detection of misbehavior actually taking place. As long as the false receiver gives reasonable answers, the sender might be fooled into thinking he is communicating with the real receiver.

Periodically sending messages with embedded X-Trace information may help to identify an attack and take the necessary countermeasures. Note, that - as elsewhere - it is important for the defender to retrieve X-Trace reports over a secure channel, or else the attacker could replace any information he captures with his own, forged information. Unfortunately, a secure channel requires some authentication by X-Trace logging hosts by which they guarantee the sender that the X-Trace information in question is really generated by them and has not been tampered with during transit. In that situation we are back to using strong cryptography - which we could have used from the start to authenticate the original messages already.

5.3 Detection of “idle” port scans

5.3.1 Problem

A port scan is a way for an attacker to find out which applications are accepting connections on a certain host. It works usually on the transport layer, using TCP or UDP ports, although other variants are possible. We will concentrate on the TCP variant here. Idle scans allow an attacker to conduct a port scan without revealing his own IP address.

During a typical TCP port scan, the attacker starts by sequentially trying to open a connection to each TCP Port he is interested in. He does so by making his computer send a "TCP-SYN" segment - that is, a TCP segment with the "SYN"-flag set in its header. The TCP protocol then dictates, that the targeted host must answer either with a "TCP-RST" segment (A TCP segment with the "RST"-flag set), if no service is listening at that port, or with a "TCP-SYN/ACK" segment (a TCP segment with both, the "SYN"-flag and the "ACK"-flag set), if there is a service on that port and the asking host may start to send data. Following the TCP protocol, the asking host should then answer with a "TCP-ACK" segment (A TCP segment with only the "ACK"-flag set), establishing the connection.

The idle scan was first mentioned in a mail by Salvatore Sanfilippo to the bugtraq mailing list [San98] and is described in detail by Gordon Lyon in [Lyo]. It is a specific kind of TCP port scan, allowing the initiator to scan a host without revealing his IP address. This of course is a very valuable property for any attacker, as it allows him to stay hidden, not revealing his identity to the defender.

This specific kind of port scan however requires the (possibly involuntary) cooperation of another host with certain properties: First, the third host - called "zombie" here[1] – is required to set predictable, alternating values as IP Fragment IDs in its IP header - typically just incrementing the field by one for each packet sent. Second, the zombie host must not get any other traffic during the scan. This should be easy to fulfill however - company workstations will be a good candidate during the night for example.

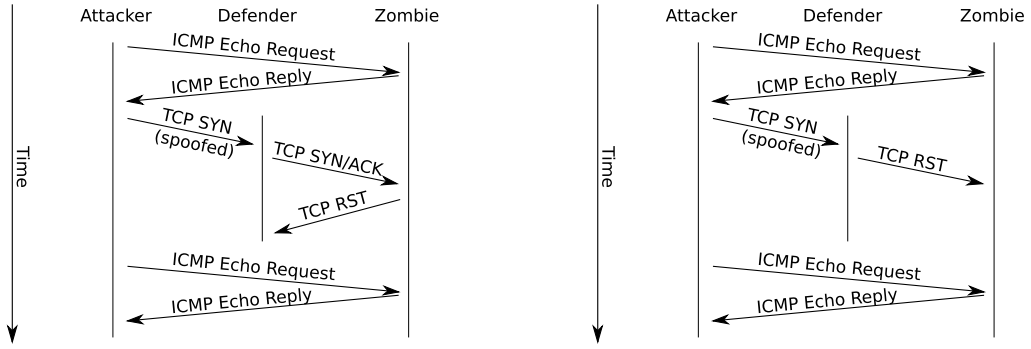
The attacker then replaces the IP source address in his "TCP-SYN" segments with the public address of the zombie. From the defender point of view it now appears as if the zombie host was carrying out a port scan. Based on the IP source address alone, the defender is unable to tell an idle scan with the zombie host as cooperating host apart from a common port scan originating at the zombie host. Thus, the attacker achieved his goal, tricking the defender into thinking he was attacked by the zombie instead of the attacker.

There remains however one problem for the attacker: he does not get the results of the connection attempt as the answers are routed to the zombie host. This is, where the special property of the zombie host comes in: The attacker sends an IP packet that causes a reply to the zombie host. A convenient way to do so is to use the Internet Control Message Protocol (ICMP) to send an "Echo request" packet which is answered by the receiving host with an "Echo reply" packet. By pinging the zombie host before and after attempting to connect to the defender, the attacker is able to determine whether the zombie host sent an IP packet in the meantime by comparing the IP fragmentation ID fields of the two replies he got. The zombie host on the other hand, will react to an unexpected "TCP-SYN/ACK" segment with a "TCP-RST" segment, while simply dropping any unexpected "TCP-RST" segments.

Now the attack is carried out as follows (see figure 3):

1. The attacker pings the zombie host to get its current IP fragmentation ID
2. The attacker sends a "TCP-SYN" segment to the defender, replacing the IP source address with the public IP address of the zombie host
3. The defender replies to the zombie host with either a "TCP-SYN/ACK" segment (if the port was open) or a "TCP-RST" segment (if the port was closed).

[1] Note that, in this context, "zombie" implies a node that is unvoluntarily contributing to an ongoing attack without explicitely being infected by malware



(a) Open port (b) Closed port

Figure 3: Packet exchange during an TCP idle scan: 3(a) with an open port; 3(b) with a closed port

4. The zombie host either responds with a "TCP-RST" segment (if he got an unexpected "TCP-SYN/ACK" segment), using the next IP fragmentation ID in this packet, or he drops the packet (if he got an unexpected "TCP-RST" segment), leaving the IP fragmentation ID sequence as is.

5. Meanwhile, the attacker waits some time to allow for the answer to arrive at the zombie, then pings the zombie again to get its current IP fragmentation ID. If it is the next number in the sequence, the attacker concludes that the zombie host did not send a packet and the port was closed. If the IP fragmentation ID is the number after the expected next number in the sequence, the attacker concludes that the zombie host sent an IP packet - presumably a "TCP-RST" segment upon an unexpected "TCP-SYN/ACK" segment - and that the port is open.

There are several things to note: First, the zombie host is not required to increment the IP fragmentation ID field by one for each packet sent for this method to work. The exact requirement is that there is one sequence for all packets (instead of an individual sequence for each host) with alternating values (i.e. not always the same value) and that the attacker is able to predict that sequence. This requirement is trivially fulfilled by a host that increments the IP fragmentation ID for each IP packet that is sent. It is however also fulfilled if this field is set by a Pseudo Random Number Generator (PRNG), provided that the attacker is able to predict the next number in the sequence - a problem that became apparent recently, when flaws were discovered in the OpenBSD PRNG [Kle07], leaving several applications and operating systems open to prediction of numbers in the sequence.

Second, the attacker is not required to use ICMP echo request packets to retrieve the current IP fragmentation ID from the zombie host. In fact, any IP packet will do as long as it triggers a reply from the zombie host. In particular, the attacker can send a "TCP SYN/ACK" segment and wait for a "TCP RST" segment from the zombie host. At this point, attacker and defender are indistinguishable from the zombie hosts point of view - both sides keep sending unexpected "TCP SYN/ACK" segments.

Finally, the port scan reveals to the attacker all reachable ports as seen from the zombie host. This may allow the attacker to bypass a firewall that filters packets based on IP source addresses. Using this method, the attacker can then in a situation where he cannot attack his

real target, due to firewall protection, try to find a zombie host whose traffic to the real target is not filtered by a firewall, try to capture that zombie host and stage his attack on the final target from that host.

Here will be discussed how X-Trace can be used to:

1. Rule out the zombie host as the source of the attack.
2. Narrow down the location of the attacker.

The hosts involved in an idle scan can be categorized into four parties:

- The attacker
- The defender
- The zombie host
- Any (third-party) router connecting the three

Each of these parties has a different view on what is happening: The attacker obviously knows exactly what's happening - after all he is the (direct or indirect) source of all network traffic. The defender only sees incoming packets, carrying the IP source address of the zombie host, concluding that - probably - the zombie host is trying to conduct a port scan on him. The zombie host, on the other hand, simply gets unexpected traffic from two different hosts - the attacker and the defender, responding as necessary. Finally, the routers are forwarding single IP packets without a clear idea of their relationship.

While the defender is unable to uncover the attacker's identity alone, both the zombie host and - under certain circumstances - some of the involved routers have the necessary information at hand to identify the attacker.

The zombie host, once he realizes that an idle scan is going on, knows that either the defender or the attacker is the one using him as relay. He is getting network traffic from only those two parties, either getting only ICMP echo requests from the attacker and unexpected "TCP SYN/ACK" segments from the defender (in which case he already has the information necessary to conclude that the attacker is the one conducting the idle scan) or getting a lot of unexpected "TCP SYN/ACK" segments from the attacker and some unexpected "TCP SYN/ACK" segments along with a lot of "TCP RST" segments from the defender (making it a bit harder, but still not impossible, for him to conclude that the Attacker is the real culprit).

5.3.2 Solution

Solving this problem requires several steps to be taken by the defender and - depending on the situation - one step to be taken by the zombie host.

- The defender has to detect the ongoing port scan.
- Next, the defender needs to realize that incoming packets are carrying a spoofed source address and that the zombie host is not the one scanning him.
- Also, the zombie host may have to realize he is being used as an involuntary participant in an idle scan. Once he realizes the situation, he will have to start X-Trace logging on his packets, trying to prove his innocence.

- Finally, by combining the X-Trace information generated by several different hosts, the defender will have to try to reveal the real identity of the attacker.

Detecting the port scan How to detect an ongoing port scan is outside of the scope of this document. Several methods of port scan detection have been discussed elsewhere [JPBB04, SXEK05]. It will be assumed that the defender somehow already suspects a port scan being conducted on his machine.

Detecting incoming spoofed packets There are two ways for the defender to realize that incoming packets carry a spoofed IP source address. On the one hand, the defender may compare the hop-count field of incoming IP packets with previously saved values, as described in [HW07].

On the other hand, if that method does not carry the desired results (for example because the attacker has the same hop distance from the defender as the zombie host), the defender has to rely on X-Trace information generated by intermediary routers. In particular, if the packets were not processed by the access router of the zombie host the attacker knows that the IP source addresses have been spoofed.

Alerting the zombie host Basically the zombie host could suspect an ongoing idle scan using him as involuntary participant. There will be few other situations in which two seemingly independent communication partners will contact him in turn, at least one side with a lot of unexpected TCP segments.

However, decision of when an idle scan is taking place would be based on probabilities and require the zombie host to run a detection software, tracking several network packets - all without being really involved himself. Instead a better solution may be to simply enable X-Trace logging for a few packets as soon as a packet with X-Trace metadata is received.

This will allow the defender to initiate X-Trace logging on the zombie host by embedding X-Trace metadata within the "TCP SYN/ACK" or "TCP RST" segments he is sending to the zombie host. Later on, the defender can then collect X-Trace reports from within a certain time frame from the zombie host.

Note, that the zombie host is expected to receive traffic only from the attacker and the defender (therefore, the "idle" in "idle scan" - otherwise this port scan method would not work for the attacker). It follows, that all packets sent from the zombie host that do not have the defender as destination, must have the attacker as destination. Thereby, the analysis of the collected reports will uncover the identity of the attacker, solely by cooperation between defender and zombie host.

Revealing the location of the attacker If intermediary routers are all X-Trace aware, the defender is able - by reverse tracing the attackers requests - to identify the attacker. In particular, the router immediately next to the attacker will point to the attacker as the one who issued the packets in question.

If reverse tracing does not work for the defender, but he is already convinced, that the zombie host is not the one conducting the port scan, he may combine his own information with the X-Trace information of the zombie host to close down on the identity of the attacker.

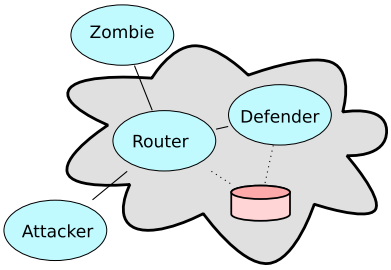


Figure 4: Single router

Having additional X-Trace information of other routers on the path between the attacker and the zombie host, the defender is then able to reduce the number of suspects to only two.

5.3.3 Simulation

Overview Simulation was done by creating several virtual machines (qemu), networked with each other, using hardcoded routing tables. All hosts in the scenario are X-Trace aware, logging their information to a central X-Trace database, apart from the attacker, which does not generate any reports. Access to that database is not restricted - any host may ask for any information.

It has also been assumed, that only one attacker (controlling only one host) is present in the network - all other hosts - routers and edge hosts - are considered to be trusted third parties. Under these assumptions, several different scenarios have been studied.

Single router - one hop In this setup, attacker, defender and zombie are connected via a single router that processes all traffic sent between the three parties (see figure 4). All three parties - attacker, defender and zombie - are equidistant in this situation, with regard to the hop count, making detection of spoofed IP packets via hop count measuring impossible.

If the intermediate router is not X-Trace aware in this scenario, the defender has no way of realizing that the zombie host and the attacker are different entities. He can trust neither of them and he is not going to get any additional data from the router. Even if the attacker is forced to initiate X-Trace logging, his X-Trace reports are obviously unreliable from the defenders point of view.

While the zombie host could try to relieve himself from suspicion by telling the defender he was pinged by the attacker, he will have a hard time convincing the defender that he says the truth. Even if the zombie provides X-Trace data on the ICMP echo replies he sends to the attacker, that data is useless, as the central router will not acknowledge it (not being X-Trace aware) and all other involved hosts (only the attacker and the zombie) are not trustworthy. And even if the central router would be logging X-Trace data, that data would still be useless as the zombie host could simply be sending out unrequested ICMP echo replies to a random host to "prove" his innocence. Only if the zombie host requires the attacker to enable X-Trace logging on his ICMP echo requests, he will be able to convince the defender of his innocence, still provided the intermediate router is X-Trace aware, resulting in all involved parties having to be X-Trace aware: the defender (for getting the reports, the attacker (for initiating X-Trace), the zombie (forcing the attacker to use it) and the intermediary router (for acknowledging the existence of an ICMP echo exchange between the attacker and the zombie host before and

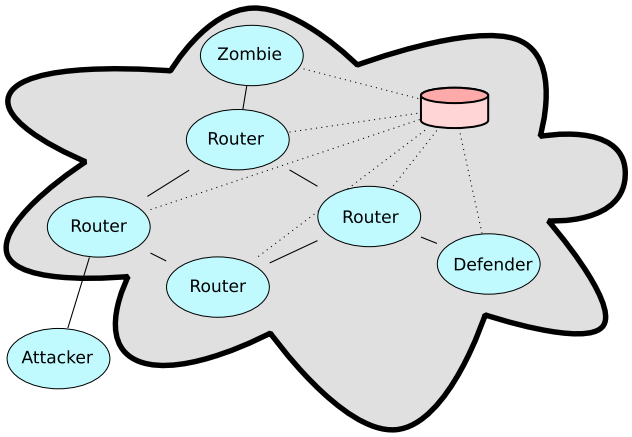


Figure 5: Multiple routers, multiple hops

after the attackers TCP connection attempt).

Yet another problem arises: in such a scenario detecting spoofed traffic by hop count does not work for the defender, as both the zombie host and the attacker are the same number of hops away. The defender has to rely on other means to detect a spoofed attack.

Now - considering a X-Trace aware router - what are the options for the defender?

In the case of the defender setting up a firewall, that only allows traffic with embedded X-Trace information, the attacker is forced to provide at least minimal X-Trace information (though his own reports may be deliberately false). This however also causes the router to generate his own X-Trace report, stating where the packet came from and where it was routed to (from the attacker, to the defender).

When the defender sends his response (either a "TCP-SYN/ACK" or a "TCP-RST") to the zombie, he just has to reuse the X-Trace ID of the incoming request, signifying a causal relationship between request and response. The router will then again create his report, stating that this time the packet was routed from the defender to the zombie.

As soon as the defender collects the reports of the router, he can easily realize that the request was not originated by the zombie, but by the attacker.

Multiple routers - multiple hops In this scenario attacker, defender and zombie are separated by several routers, providing for different hop counts between each party (see figure 5).

It will be very hard for the attacker to find out the correct hop distance between the zombie host and the defender, without having intricate knowledge about the network structure. By monitoring false hop counts, the defender can therefore detect an incoming idle scan and conclude, that the zombie is probably innocent.

From the zombie hosts point of view, the real attacker is easily visible. Since the zombie host only gets traffic by the defender and the attacker (otherwise the idle scan wouldn't work anyway), once he realizes he is part of an idle scan, he can conclude that the attacker is the

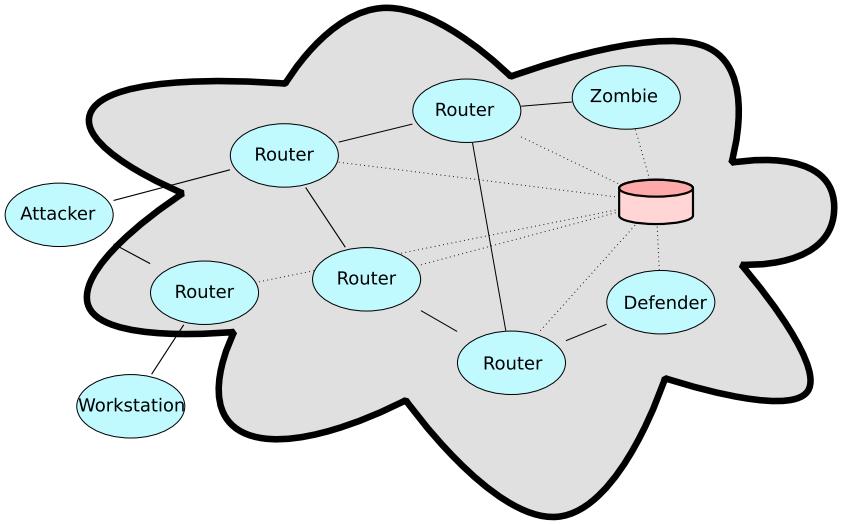


Figure 6: Idle scan with spoofed pings

one, that is pinging him. Provided the zombie host is X-Trace aware, he could then tag the next "ICMP Echo reply" with the last X-Trace ID he got from the defenders "TCP-SYN/ACK" or "TCP-RST" packet. However a better idea, as discussed in 5.3.2, will be to wait for the defender to send X-Trace enabled packets and then enabling X-Trace logging on all sent packets for a certain time.

This allows the defender to collect the generated information later on and to conclude from the existence of traffic between the zombie host and the attacker, that the attacker was the one carrying out the idle scan.

Things get more interesting though, if the attacker predicts this behavior. In that case, he may carry out a conventional port scan, trying to convince the defender that he is the zombie in an idle scan. To do so, he starts with a maximum "IP TTL" value that differs significantly from what his operating system would usually choose.

Furthermore he would have to send unrequested "ICMP Echo reply" packets to some other, random host, accusing that host of using him as zombie host in an idle scan. That behavior is, however, revealed by X-Trace enabled routers along the path. Their reports will clearly indicate, that the TCP requests originated at the attacker, not some other host.

Idle scan with spoofed pings There is an even more interesting scenario to look at. If the attacker is able to get control on a router, he may try to fool the zombie host by also spoofing the IP source address of all packets sent to the zombie host. This requires the attacker, however, to be able to intercept all traffic between the zombie host and the spoofed IP address - a requirement easily managed, if the attacker controls a router that has to be passed when sending a packet from the spoofed IP to the zombie host. Good targets for the attacker would be the access routers of either the zombie host or the host corresponding to the spoofed IP address (which we will call "spoofed source host" here).

In that case, the attacker may even be able to provide a valid hop count in his packets -

Scenario	Attack with MD	Attack without MD
Single router - one hop	Reverse tracing	Impossible to detect
Mult. routers - mult. hops	Reverse tracing	via Zombie host
With spoofed pings	Reverse tracing	via Zombie host and last Router

Table 2: Methods of uncovering the attacker during an idle scan

by sending an "ICMP Echo request" to the spoofed source host, he will readily tell him the hop count distance via the TTL value in his "ICMP Echo reply". The attacker simply has to take the TTL value provided, subtract one and use that value in all packets sent to the zombie host.

Assuming that all routers in the scenario are X-Trace enabled, the attacker still can be located as one of two hosts in the scenario. The defender has the option of either retracing the incoming TCP requests to the attacker or following the zombie hosts "ICMP Echo reply" packets. In either way, the last router before the attacker will report packets as received by / delivered to the attacker.

The attacker now has two options: either to generate a forged X-Trace report, pointing to the router between him and the spoofed source host. However, that router does not generate any X-Trace information - a fact that will be apparent to the defender as soon as he requests the information.
Also, the attacker may simply refuse to generate an X-Trace report - in that case, the defender will notice the missing report at the attacker.

In any case, the defender knows now, that either the last host providing a report or the first host not providing a report is the host responsible for the idle scan. If both routers adjacent to the attacker are considered trustworthy, the defender can even safely assume that the attacker is indeed the attacker.

Implementation To research the effects of using X-Trace information during an idle scan, a custom implementation of an idle scanner has been created first, that will - upon request - insert X-Trace metadata in its connection attempts.

Using the aforementioned assumptions - all hosts trusted except the attacker; one central X-Trace database - the defender was able to indicate the attacker in real time in all scenarios, implementing the pseudo-code showed in listing 1.

The defender followed the methods shown in table 2. In all scenarios, if the attacker provided X-Trace metadata, the defender used the generated X-Trace reports to immediately trace the incoming packets back to the attacker. In the "Single router - one hop" scenario, detection via the zombie host is impossible if the attacker can not be forced to provide X-Trace metadata, as the defender has no way of knowing, that the incoming packets are spoofed. In the "Multiple router - multiple hops" scenario the defender recognizes the incoming packets as spoofed by comparing them to his internal hop count table. He then identifies the attacker by following the X-Trace reports that are generated between the zombie host and the attacker. The "Spoofed ping" scenario is more or less similar to the "Multiple router - multiple hops" scenario, with the exception, that the X-Trace reports generated between the zombie host and

```
if (packet has X-Trace metadata) {
    retrieve relevant X-Trace reports
    attacker is last host indicated
} else {
    if (hop count does not match internal table) {
        assume spoofed IP
        send response with embedded X-Trace metadata
        wait for next packet to arrive
        collect reports
        filter reports:
            source host is zombie
            AND time is between last two packets,
            AND destination host is not self
        attacker is last host indicated
    } else {
        Either:
            attacker has same hop count as zombie
            OR this is real scan
    }
}
```

Listing 1: Pseudo-code for locating an idle scanning attacker with the help of X-Trace

the attacker have to be followed carefully - it does not suffice to simply identify the IP address the zombie host is sending his packets to as the attacker.

5.3.4 Issues

There still remain some situations in which the attacker can not be determined reliably. In particular, if the attacker either has the same distance (with regard to the hop count) from the defender as the zombie host, or is able to guess the hop count distance between defender and zombie host and if he can not be forced to embed X-Trace metadata into his packets, the defender will have a hard time finding out, who really attacked him.

A possible way for the attacker to guess the correct hop count between zombie host and defender opens up in a fully X-Trace enabled scenario. In such a scenario, the attacker could, before carrying out his idle scan, send a packet to the zombie host that contains X-Trace metadata and the IP address of the defender as spoofed source address. By analyzing the X-Trace reports generated on the path, the attacker is then able to infer the correct hop count distance between the zombie host and the defender. However, this causes X-Trace reports to also be generated on the path between attacker and zombie. By inspecting the X-Trace information embedded in the incoming packet and requesting the corresponding reports, the defender will still get a pointer to the attacker - provided that the reports are saved long enough for the defender to recognize the following attack and decide to take a closer look at all packets he received before the attack.

Another difficulty for the defender is a zombie host, that does have a low amount of traffic with other, uninvolved hosts. A certain low amount of "background noise" traffic is acceptable for the attacker during the idle scan, causing him only to scan each port several times until

he can be reasonably sure that his results are correct. The difficulty for the defender lies in distinguishing the attacker from other uninvolved hosts the zombie host communicated with. Still, the amount of traffic generated by the attacker at the zombie host will easily surpass the amount of traffic the zombie host can have with other parties during the scan - a fact that still may be a hint for the defender to come to the right conclusion.

Finally, the defender has to rely on the cooperation of other hosts that will usually not be under his control. If no other host will generate X-Trace reports, or X-Trace reports are generated, but the defender has no way of accessing them, all efforts are futile - the defender has no way of inferring the correct attacker from the data remaining to him.

6 Conclusion

In this deliverable, we have discussed how to use information provided by distributed correlated monitoring tools in order to detect and analyze challenges in computer networks. It has been shown that this information is able to uncover the source of challenges in scenarios where it would have been difficult to diagnose the fault with conventional isolated monitoring systems. This effect has been illustrated in three different scenarios. In these scenarios it was, in particular, the ability to correlate information from distributed points in the network that allows to correctly diagnose the challenge. We conclude that distributed correlated network monitoring should be considered as an important tool to improve the resilience of a given network.

References

[BBF+10] Radovan Bruncak, Nafeesa Bohra, Andreas Fischer, Steven Simpson, Justin P. Rohrer, James P.G. Sterbenz, David Hutchison, and Hermann de Meer. Resumenet deliverable d1.4: Cross-layer optimization and multilevel resilience, Aug. 2010.

[BDKC10] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proc. of the 10th annual conference on Internet Measurement (IMC)*, pages 206–217, 2010.

[DDK+10] B. Delosme, C. Dousson, N. Kheir, C. Lac, and P. Smith. Service surveillance and detection of challenging situation, Jul. 2010.

[EBRR11] M. Errais, M. Bellafkih, B. Raouyane, and M. Ramdani. Distributed network monitoring for ims network. In *International Conference on Multimedia Computing and Systems (ICMCS)*, pages 1–6, 2011.

[FD10] F. Fusco and L. Deri. High speed network traffic analysis with commodity multicore systems. In *Proc. of the 10th annual conference on Internet measurement (IMC)*, pages 218–224, 2010.

[HKB04] A. Habib, M. Khan, and B. Bhargava. Edge-to-edge measurement-based distributed network monitoring. *Computer Networks*, 44:211–233, 2004.

[HW07] Kang G. Shin Haining Wang, Cheng Jin. Defense against spoofed ip traffic using hop-count filtering. In *IEEE/ACM Transactions On Networking*, volume 15, Feb. 2007.

[JPBB04] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, May 2004.

[Kle07] Amit Klein. Openbsd dns cache poisoning and multiple o/s predictable ip id vulnerability. Technical report, Trusteer, 2007.

[Lyo] Gordon "Fyodor" Lyon. Idle scanning and related ipid games. http://nmap.org/idlescan.html[1].

[MGM06] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, March 2006.

[San98] Salvatore Sanfilippo. "new tcp scan method", Dec. 1998. http://seclists.org/bugtraq/1998/Dec/0079.html[1].

[SFM+10] P. Smith, M. Fry, S. Martin, L. Chiarello, M. Fischer, Ch. Rohner, G. Popa, H. De Meer, A. Fischer, and N. Bohra. New challenge detection approaches, Aug. 2010.

[SXEK05] G. Simon, H. Xiong, E. Eilertson, and V. Kumar. Scan detection: A data mining approach. Technical report, University of Minnesota, 2005.

[VC11] Solange Rito Lima Vasco Castro, Paulo Carvalho. Toward a scalable and collaborative network monitoring overlay. In *Third COST TMA International Workshop on Traffic Monitoring and Analysis*, 2011.