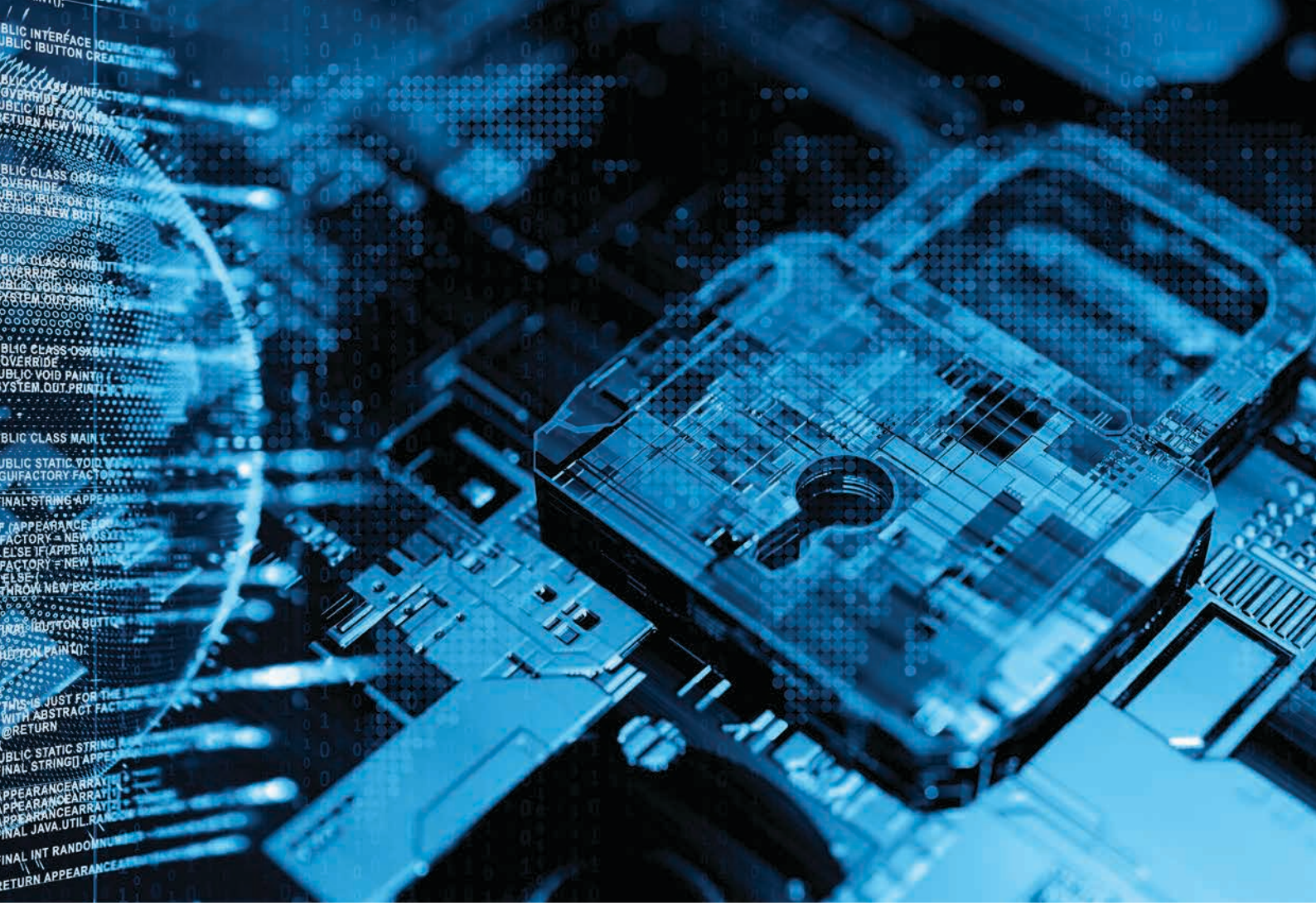

Technology explained

Protecting confidental information using datadiodes

White Paper by
dr. Wouter Teepe, for Fox-IT
and Colin Robbins, for Nexor

FOX IT
part of nccgroup

fox-it.com/datadiode

Introduction

When protecting an isolated network against outsider attacks, there are a number of objectives and technologies that are commonly used. Objectives typically boil down to C.I.A.: confidentiality, integrity and availability. The best possible technology for confidentiality is the unidirectional network connection by means of a datadiode. However, there is a lot of technology relating to datadiodes that impacts integrity and availability. In particular, protocol breaks and content checking have a subtle relation to these objectives. This briefing paper will explain how these technologies relate to one another and to the principal C.I.A. security objectives.

This paper focuses on situations where confidentiality has priority over integrity, where 'protecting secrets' (Figure 1) is essential. Datadiodes can also be deployed for 'protecting assets' (Figure 2), where integrity is essential and confidentiality is of secondary priority, typically when protecting industrial installations. For the sake of clarity, we will focus on the 'protecting secrets' scenarios in this paper.

Protecting secrets

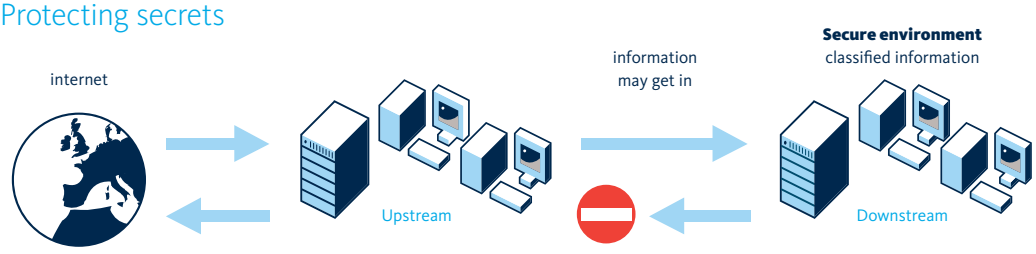

Figure 1: the protecting secrets scenario

Protecting assets

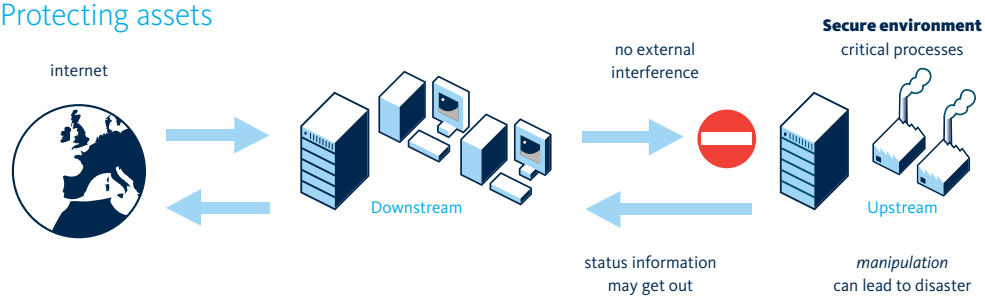

Figure 2: the protecting assets scenario

Unidirectional network connections

A unidirectional network connection is a link between two networks for which it can be guaranteed that the information only flows from the one network to the other, and not in the other direction. The source network is typically referred to as 'upstream' and the destination network as 'downstream'. A typical scenario is where the downstream network contains highly classified information which should not be leaked to the outside world, while the upstream network is directly or indirectly linked to that outside world. In this scenario, the unidirectional network connection is 'protecting secrets'. The unidirectional network connection prevents 'data leakage' or 'data exfiltration' from the downstream network.

However, confidentiality may not be the only protection objective of the downstream network. Due to the uni-directional network connection, data cannot come out of the downstream network, but the data flowing into the downstream network may still cause harm. The data could 'attack' the downstream network.

A unidirectional network connection is often implemented and enforced using a network device called a datadiode, as described in this paper, supported by specialist security software.

Creating a unidirectional network connection does not prevent all methods of data leakage.

1. If there is another network connection between the upstream and downstream network next to the datadiode, data can be exfiltrated by means of the other network connection.
2. If it is possible on the downstream network to store information on portable storage media such as USB sticks, this media can be physically exfiltrated and as such provide a means for data leakage. Controls are needed to prevent people from using such media. Controls may vary from technical controls such as disabling USB ports to procedural controls such as complete prohibition of portable media. When a datadiode is deployed, the operational benefit of using such media decreases so much that it is possible to impose strict business policies on portable media use without a major business impact. Without a datadiode, a strict portable media policy would lead to highly impractical situations.
3. A datadiode does not prevent people from printing documents and carrying them to places where they should not end up, or from reading documents and telling the contents to people who are not allowed to know it.

Attacking the downstream network

Computer security attacks come in many forms; a common method of attack is to get a computer to behave in a way not considered by the designers and seek to take advantage of that. Modifying protocols, for example, to send information that is non-compliant to the protocol is one way of inducing errors in a poorly designed system.

A common example of this kind of attack is the buffer overflow[1]. Buffer overflows can occur at any layer in the protocol stack – from the network interface to the application. Buffer overflow vulnerabilities have been seen in all kinds of places, ranging from PDF files[2] to network cards[3].

So when letting data onto a downstream network, the network may be exposed to attacks which are embedded into the information flowing onto that network, even if the information flow is one way. A datadiode makes sure that such an attack cannot lead to data leakage – even if the attackers manage to establish a command and control server, the server will not be able to communicate back via the datadiode. However, availability and integrity of the downstream network are potentially still at risk.

This is almost where the protocol break enters the arena. However, let us first look closely at the information flowing into the downstream network.

In general, this data can be divided into payload data and traffic control data. The payload data contains the data that the sender wants to send to the downstream network. For example, this may be a file, an email or a print job. This payload data is essentially static: the message that is sent should be the same as the message that is delivered (later in the paper we discuss some security reasons why the message may be deliberately transformed into something else). The payload may also contain complex types with multiple files embedded such as ZIP and MIME formats.

Figure 3: data send using a protocol

1 https://en.wikipedia.org/wiki/Buffer_overflow
2 http://resources.infosecinstitute.com/hacking-pdf-part-2/
3 http://theinvisiblethings.blogspot.co.uk/2010/04/remotely-attacking network-cards-or-why.html

Protocol

To deliver the payload, a *protocol* is used. A protocol is a set of communication agreements, which ensure that if both sides of a communication channel adhere to it, the payload gets delivered correctly. To achieve its design objectives, a protocol introduces extra data into the data flow to coordinate these protocol specific goals: traffic control data. A protocol takes care of many things that a normal computer user is never aware of: that the payload gets routed in the right direction; that it is chopped into parts where needed and reassembled again where possible. Protocols may do very complicated things like compression, tunneling, load balancing, authentication, caching, spooling, all kinds of things to make the communication go smoothly. Examples include FTP, SMTP and HTTP.

All this complexity which goes into these protocols makes the system work, but only under the condition that both sides are cooperative. An attacker may take the liberty not to be cooperative, and send malformed *traffic control* data. This can cause a buffer overflow or other fault in the receiving system, and with it launch a successful attack. Heartbleed[4] was an example of this where the attacker chose not to be cooperative by misinforming the protocol about the size of the payload.

In the 'protecting secrets' scenario it can generally be assumed that the attacker has access to the upstream network. From the upstream network, the attacker could attack the downstream network by abusing a design flaw in one of the systems on the downstream network.

A unidirectional network connection prevents such an attack from leading to data leakage. The attack may still cause harm in terms of integrity and availability on the downstream network. A protocol break effectively cuts out attack vectors which live in the traffic control data, as will be discussed next.

4 http://heartbleed.com

Protocol Break

The attacks that can be caused by one of the parties not adhering to a protocol can only be prevented by making sure that in the environment where attacks are not acceptable, both parties in the protocol are trusted. For unidirectional communication scenarios, that means that the side sending the payload (upstream) should be trustworthy, at least from the perspective of the receiver (downstream). The only way to ensure this is by the application of a protocol break.

A protocol break consists of two components that sit between the sender and the receiver of a message. The first one is a 'catcher', which, while adhering to the protocol, strips all *traffic control* data from the data it receives, and keeps only the *payload* data. The second component is a 'thrower'. The thrower does the opposite: it takes bare payload data, and sends the payload to another system by means of some chosen protocol. In order to do this successfully, the thrower does all the complicated things that are necessary to adhere to the protocol specifications, including the creation of *traffic control* data.

The diagram shows that system B (downstream) will never directly speak to system A (upstream) – communications go via the catcher and thrower. This means that an attacker must undergo a long chain of attacks to reach system B. At first glance, you might conclude that the catcher and the thrower only make the attack on system B somewhat more cumbersome but not impossible. *There is an ingenious way of preventing this.*

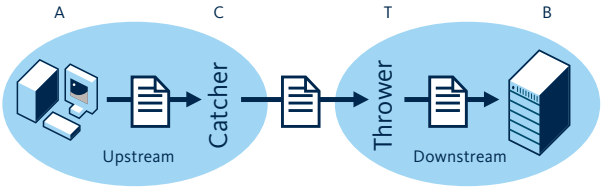

Figure 4: a protocol break

Protocol Break in action

What does the complete chain look like?

1. System A wants to send a message to system B.
2. The traffic from A towards B is routed to C, the catcher. System A may believe it is talking to system B, but in fact it is talking to a catcher, which acts as a proxy for system B. Systems A and C exchange data via means of a protocol. They exchange both *traffic control* data and payload data.
3. System C distils a payload and provides this *payload* to system T, the thrower.
4. The thrower collects the payload data, and sends it to system B via means of a protocol. System B may believe it is talking to system A, but in fact it is talking to a thrower, which acts as a proxy for system A. Systems T and B exchange both *traffic control data and payload* data.

Datadiode Protocol

As we have already discussed, protocols may be very complex. They may be implemented in hardware or software; these implementations may have slight design flaws which permit an attack on the system by exploiting these flaws. Basically, this is the case because these protocols are often designed and implemented for a specific function, where security was not considered essential, or of secondary interest in best case.

In the system described in Figure 4, the catcher and thrower need some means of communication as well – they need a protocol between them. This protocol is under the full control of the security system designers. Security of the protocol can be made a top design priority. The protocol can be designed in such a way that the complete state space of both catcher and thrower can be analyzed. The protocol can be designed in such a way that it is very trivial to detect any protocol deviations that may be malicious. This special protocol between the catcher and the thrower is called the *datadiode protocol*.

How does the datadiode protocol guarantee that when the catcher system C is corrupted, the thrower system T remains unaffected? Assume the protocol between the attacker system A and the catcher system C is very complicated. The attacker might be able to find an exploit in the catcher, attack it and corrupt it. However, the catcher cannot be used as a stepping stone to attack the thrower. The datadiode protocol spoken between the catcher and the thrower leaves no room for attacking the thrower because of its design. Were the catcher to try this, the thrower would detect malformed protocol data and simply ignore it. The important observation here is that because the thrower cannot be corrupted, the thrower will remain secure and cannot act as a stepping stone to attack system B.

How does a protocol break work together with a datadiode? The datadiode is put exactly between the catcher and the thrower. The catcher and the thrower are often referred to as 'proxy servers'. The catcher resides in the upstream network, and is often referred to as the upstream proxy server. The thrower resides in the downstream network and is often referred to as the downstream proxy server.

Vulnerability assessment

Which components of the system are vulnerable for attacks by means of deliberate protocol deviations?

- First: System B. However, B only talks to system T. System T is not under control of an attacker and as such it cannot be attacked from system T.
- Second: System T. System T talks to system B which is supposed to be clean and to system C. System C is not initially under control of the attacker, but might be exploited.
- Third: System C. System C is initially trusted, but talks to system A, which may be under the control of an attacker. This may seem to be a problem, but it is not.

Payload and content checking

We have not looked at the payload data yet. Using a datadiode and a catcher and a thrower, we have assured that the delivery process of the payload cannot inflict harm, and that the payload, if malicious, cannot exfiltrate data. However, the legitimacy or non-maliciousness of the payload itself has not been verified. This is a fundamentally complicated issue. The payload, for example a PDF file, may be constructed in such a way that the software which presumably will be used for viewing the PDF file, may be exploited. To address this, there are fundamentally five approaches which are shown alongside.

Naturally, which approach is acceptable from a function, cost and risk perspective will differ from case to case. There is no magic bullet here, and depending on your adversary, risk appetite, functional requirements and budget you may choose your own balance between these approaches.

How to prevent exploitation by the payload

1. Accept the risk. After all, the downstream network is completely isolated.
2. Do a very strict pattern matching on the payload, only accepting payloads recognized to be conformant (i.e. whitelisting). For example, only accept 'text' files with 7 bit ASCII characters in it.
3. Do some pattern matching on the payload, removing payloads recognized to be wrong (i.e. blacklisting). This is essentially what an anti-virus solution does. It keeps a lot of bad things out, but it gives no guarantee whatsoever. An advanced persistent threat will be capable of ensuring its malicious payload will not be recognized by using a zero-day exploit.
4. Convert the payload itself. Essentially, take all information out of the source file, and create a new one with the same contents. Conceptually this is the same as what the catcher and the thrower do, but now at the payload level. There is no general solution for this; it turns out to be extremely complicated and only works for very well defined use cases.
5. Do a combination of the above. For example: only accept JPEG files, convert those to PNG and drop all other payloads.

Could we also do just one of both?

Can a protocol break be implemented without a datadiode? And can a datadiode be used without a protocol break? Of course, it depends. These are interesting questions.

Firstly, a protocol break without a datadiode: technically this is perfectly possible. The catcher communicates directly with the thrower without a datadiode between them. It will work. However, there are two caveats.

a. Without a datadiode, there is no strong guarantee that the information will only flow in the desired direction. All kinds of provisions will have to be designed in the software to guarantee that there is no communication channel back possible; that there is no backchannel. Fundamental research has shown that doing this in software is extremely complicated; it is a large undertaking to do this to a high level of assurance. Approaches to product evaluation, such as Common Criteria, provide some level of assurance via peer-review and testing, but the results cannot be 100% guaranteed. Only a datadiode can provide a 100% guarantee.
b. Without a datadiode, is it very difficult to establish whether the 'protocol break solution' genuinely provides a protocol break. The provider of the solution has to be trusted that it has not cut corners anywhere in the design and implementation of the solution. With a datadiode in the middle, you can be absolutely certain that there is a protocol break: without a protocol break, the setup would simply not work. Or put in a 'confusing' manner: the protocol will break when using a datadiode. There is one exception, this is when a protocol is unidirectional by design.

A protocol is unidirectional by design when all messages that are needed to make the protocol work are sent in the same direction. In particular, the sender will keep on working and sending data without ever receiving any acknowledgement. A protocol that is unidirectional by design will keep working when a datadiode is put between the sender and the receiver. The datadiode protocol in particular is such a protocol.

Secondly: a datadiode without a protocol break. This will only work for protocols that are unidirectional by design. Participants in such a protocol will not notice a datadiode between them (as long as it is connected in the right direction). However, virtually every protocol nowadays is bidirectional. Everything that uses TCP is bidirectional. The TCP protocol has provisions to stop sending data when no acknowledgements are received back from the addressee. When the addressee is situated behind a datadiode, the acknowledgements will never pass back through the datadiode, and as such prevent the TCP protocol from delivering the data. Almost all protocols that use UDP have similar provisions by means of extra traffic control data sent in the reverse direction.

The single and notable exception is a subfamily of UDP protocols that we call the unidirectional UDP protocols. In these protocols, the sender keeps on sending data even if it does not receive any confirmation whatsoever from the addressee. In general this is a strongly discouraged design practice, as it may lead to network congestion when not used very carefully. However, some CCTV streaming protocols and logging protocols use unidirectional UDP. When applied with care, this is perfectly sensible.

Unidirectional UDP protocols, as stated, will work through a datadiode. In this case, it is possible to use a datadiode without a protocol break. A protocol break is still recommended practice, though. The IP header of the UDP packet is complex and may be a means to deceive routers and switches on either side of the datadiode.

Unidirectional UDP is supported by almost all datadiode vendors. However, only some datadiode vendors provide a genuine UDP protocol break.

Confidentiality, Integrity and Availability revisited

So where does all this bring us? The datadiode prevents data leakage from the downstream network to the upstream network. The data that is sent from the upstream network to the downstream network can be divided into *traffic control* data (i.e. protocol metadata) and *payload* data. The protocol break ensures that known and unknown attacks in common and not so common protocols cannot destabilize systems in the downstream network. All traffic control data passing through the datadiode is fully controlled and designed to be easily rejectable in case of malformed messages. Thus, the systems in the downstream network cannot be attacked. Therefore, the integrity and availability of the downstream networks remains unchallenged.

The catcher, which resides in the upstream network, is potentially susceptible to an attack. In the worst case, this would cause the unidirectional traffic flow between the networks to stop (i.e. to become 'zero directional'). Thus, the availability of the network connection is the worst thing that could be compromised. That is still undesirable, but there is nothing that can be done about that. We might harden the catcher some more. However, because the attacker has access to the upstream network, he could disrupt any other component in the upstream network to make network traffic cease.

Let us end with a systematic assessment of C.I.A.: confidentiality, integrity and availability. In a 'protecting secrets' scenario, one aims to protect the downstream network.

Confidentiality of the downstream network
A unidirectional network connection or datadiode ensures the confidentiality of the information of the downstream network. It prevents data leakage or data exfiltration via the unidirectional network connection. Exfiltration methods like printing, USB sticks and other unprotected network connections are not prevented by a unidirectional network connection. However, using unidirectional network connections is a key enabler for enforcing strict policies on portable media usage to prevent such.

Integrity & availability of the downstream network
A unidirectional network connection in itself does not prevent attacks that may impact integrity and availability. Merely using a unidirectional network connection does however already mitigate the impact of an attack, because a successful attack cannot 'phone home' for instructions or to exfiltrate data.

To prevent an attack, all traffic passing the unidirectional network connection must be made harmless. This traffic can be divided into traffic control data and payload data. A protocol break neutralizes attacks that may come with the traffic control data.

To be absolutely safe, the payload data must be neutralized as well. This can be done, but it is very situation specific and only when one can predict very precisely what the form is of the data that will pass the unidirectional network connection. In general cases, one can do a best effort content check which permits a variety of data formats to flow in, at the price of not knowing whether all attacks have been stopped.

Availability of the unidirectional network connection
A unidirectional network connection is used to enable communication between two otherwise unconnected systems. The approach described places security controls around the *unidirectional network connection*, but in its own right cannot ensure the availability of the unidirectional network connection itself. We have not prevented an attacker taking control of the upstream servers and preventing them from sending data to the downstream network – we have mitigated the risks of the damage an attacker can cause in terms of protecting secrets, but not from denial of service.

To prevent a denial-of-service attack on the network connection, other security mechanisms and architectures must be used. Discussion of these methodologies is outside the scope of this paper.

Concluding remarks

As shown by this assessment, by using a datadiode in combination with a protocol break, we have ensured the best possible protection of the 'secrets' in the downstream network, using a fail-safe approach. Such approaches to protecting 'secrets' have been used in high assurance environments to protect state secrets for many years. These solutions are now readily accessible in commercial markets available to solution and security architects.

Our advice:

- Assess your information assets, and determine which are your most valuable business 'secrets', and the impact of these leaking from your organization.
- Locate these 'secrets': are these on networks that are connected to the Internet?
- Ask yourself: should they be on networks that are connected to the Internet?

About the authors

Dr. *Wouter Teepe*
Holds a Master's degree in artificial intelligence and holds a PhD in computer security from the University of Groningen.

Colin Robbins
Holds a Joint first class honours Bachelor's degree in Computer Science and Electronic Engineering from University College, London, and is a CESG Certified Professional (Lead Security & Information Risk Assessor).

Fox-IT

Fox-IT prevents, solves and mitigates the most serious threats caused by cyber attacks, data leaks, or fraud with innovative solutions for governments, defense agencies, law enforcement, critical infrastructure and banking and commercial enterprise clients worldwide. Fox-IT combines smart ideas with advanced technology to create solutions that contribute to a more secure society. We develop products and custom solutions for our clients to guarantee the safety of sensitive and critical government systems, to protect industrial networks, to defend online banking systems, and to secure confidential data.

FOX IT
part of nccgroup

fox-it.com

Fox-IT B.V.
Olof Palmestraat 6, Delft
P.O. Box 638, 2600 AP Delft
The Netherlands

T +31 (0)15 284 7999
F +31 (0)15 284 7990
fox@fox-it.com

Fox-IT is part of NCC Group.