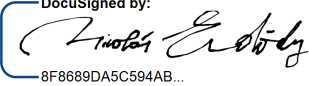




Security for the SDP Architecture Considerations

Document Number.....SDP Memo 064
Document Type.....MEMO
Revision.....01
Author.....N. Erdödy, R. O’Keefe
Release Date.....2018-08-31
Document Classification..... Unrestricted
Status..... Draft

Lead Author	Designation	Affiliation
Nicolás Erdödy	SDP Team	Open Parallel Ltd. NZA (New Zealand SKA Alliance)
Signature & Date:	 8F8689DA5C594AB...	10/21/2018 7:45:13 PM PDT

With contributions and reviews greatly appreciated from		Affiliation
Dr. Richard O'Keefe	SDP Team, NZA	University of Otago - Open Parallel (NZA)
Dr. Andrew Ensor	Director, NZA	AUT University (NZA)
Piers Harding	SDP Team, NZA	Catalyst IT (NZA)
Robert O'Brien	Systems Engineer / Security	Independent
Anonymous Reviewer	CEng (UK), CPEng (NZ)	Manager, NZ Govt

ORGANISATION DETAILS

Name	Science Data Processor Consortium
Address	Astrophysics Cavendish Laboratory JJ Thomson Avenue Cambridge CB3 0HE
Website	http://ska-sdp.org
Email	ska-sdp-pa@mrao.cam.ac.uk

1. SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Acknowledgement:

The authors wish to acknowledge the inputs, corrections and continuous support from the NZA Team Members Dr. Andrew Ensor, Peter Baillie, Piers Harding and TN Chan.

2. Table of Contents

1. SDP Memo Disclaimer

2. Table of Contents

3. List of Abbreviations

4. Introduction

5. Executive Summary

6. References

7. Four Aspects of HPC Security

7.1. High-performance by definition

7.2. Distinctive types of operation

7.3. Custom operating system stacks

7.4. Openness

7.5. Summary

8. Security and the SKA: Why it needs to be considered

8.1. Lessons learned from industry - Triton

8.2. Aren't we lucky to use Linux!

8.3. Meltdown and Spectre

8.4. But who would attack the SKA?

8.4.1. Attacks and accidents

8.4.2. Thefts of physical components

8.4.3. Thefts of resources

8.4.4. Other motivations

8.5. Important issues

Document No: 064

Revision: 01

Release Date: 2018-08-31

Unrestricted

Author: N. Erdödy

Page 3 of 37

8.5.1. User-space hardening

8.5.2. Be wary of new code

8.6. OpenStack

8.7. Findings

8.8. Recommendations

8.8.1. Draft a “less-insecure-programming” booklet

8.9. Conclusions

9. Hardware/OS Related Security

9.1. Hardware Security Module

9.2. Cloud Key Management

9.3. Virtual Trusted Platform Module

9.4. Secure Boot

9.5. Low-level vulnerabilities

9.6. Containers

9.7. Intel SGX

9.8. Keystone

10. Architecture Considerations for Security

10.1. Summary

10.2. Details

10.2.1. The Principle of Least Privilege

10.2.2. Authentication

10.2.3. Auditing / logging

10.2.4. Encapsulation

10.2.5. Encryption

10.2.6. Hardening

10.2.7. Tainting

10.2.8. Control the Environment

10.2.9. Proportionality

10.2.10. CIA - Key Concepts

11. Conclusion

12. Appendix - Debate

Document No: 064

Revision: 01

Release Date: 2018-08-31

Unrestricted

Author: N. Erdödy

Page 4 of 37

- 12.1. Security Policy for the SDP
- 12.2. Why would someone attack the SDP?
- 12.3. Is security a matter for SDP or SKAO?
- 12.4. "Security Wheel" and how CERN fights hackers
- 12.5. The impact of Meltdown & Spectre in SDP's hardware budget

3. List of Abbreviations

ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
AMD	Advanced Micro Devices, Inc.
API	Application Programming Interface
ARL	Algorithm Reference Library
ARM	Advanced RISC Machine
BIOS	Basic Input/Output System
BOINC	Berkeley Open Infrastructure for Network Computing
C&C	Component & Connector
CERN	<i>Conseil Européen pour la Recherche Nucléaire</i> (European Organization for Nuclear Research)
CIA	Confidentiality, Integrity and Availability
CI/CD	Continuous Integration/Continuous Delivery
CLMUL	Carry-less Multiplication
CNK	Compute Node Kernel
CNL	Compute Node Linux
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures

CWE	Common Weakness Enumeration
DALiuGE	Data Activated Liu Graph Engine
DDoS	Distributed Denial of Service
DMZ	Demilitarized Zone
DoS	Denial-of-Service attack
EDR	Endpoint Detection and Response
FIPS 140-2	Federal Information Processing Standard
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GNU	GNU's Not UNIX!
GPU	Graphic Processing Unit
HPC	High Performance Computing
HPE	Hewlett Packard Enterprise
HSM	Hardware Security Module
IBM	International Business Machines, Inc.
IDL	Interactive Data Language
IME	Intel Management Engine
I/O	Input/Output
IP	Internet Protocol
ITU	International Telecommunication Union
JIT	Just-In-Time
JSON	JavaScript Object Notation
KMS	Key Management Service
KPTI	Kernel page-table isolation
L1TF	L1 Terminal Fault
LGPL	GNU Lesser General Public License
MIT	Massachusetts Institute of Technology

NB	Nota bene
NERSC	National Energy Research Scientific Computing Center
NSA	National Security Agency
NvME FIO	Non-volatile Memory Express Flexible I/O
NvMEoF	Non-volatile Memory Express over Fabrics
NZA	New Zealand SKA Alliance
OS	Operating System
PEP	Python Enhancement Proposal
PKI	Public Key Infrastructure
PHP	Personal Home Page (now PHP: Hypertext Preprocessor)
POSIX	Portable Operating System Interface
RAT	Remotely Accessible Trojan
RHEL	Red Hat Enterprise Linux
RISC-V	Reduced Instruction Set Computer (“risk-five”)
ROM	Read-only Memory
RPC	Remote Procedure Call
SABSA	Sherwood Applied Business Security Architecture
SCADA	Supervisory Control And Data Acquisition
SDLC	Software Development Life Cycle
SDP	Science Data Processor
SGX	Intel Software Guard Extensions
SKA	Square Kilometre Array
SMB	Server Message Block
SMM	System Management Mode
SNMP	Simple Network Management Protocol
SPARC	Scalable Processor Architecture
SRC	SKA Regional Centre

SQL	Structured Query Language
SSH	Secure Shell
SUSE	Software und System Entwicklung
TCB	Trusted Computing Base
TEE	Trusted Execution Environments
TM	Telescope Manager
TPM	Trusted Platform Module
TLS	Transport Layer Security
UEFI	Unified Extensible Firmware Interface
VM	Virtual Machine
vTPM	Virtualising the Trusted Platform Module
WAN	Wide Area Network

4. Introduction

This document proposes a view of the architecture from a security perspective. The SDP is a system which must be secure; and which can provide a continuous service to the SKA Observatory, not to be open to the ends of attackers. Unfortunately it is virtually certain that the SDP *will* be attacked and extremely likely that at least one attack during its lifetime will be successful.

The SDP System-level Security View (AD01) states that “SDP interacts mostly with other systems, with little direct user interaction outside of the system, with the exception of telescope operators. We have to place some trust in these other systems being secure to avoid excessive operational overheads.”

“We need to invest in edge security, as SDP might become a target for attack due to being a large HPC installation with a large WAN connection that could be used for a DoS attack against other systems with access to the WAN network.”

“We assume that the WAN will be engineered so that it is only accessible from a small number of hosts worldwide to limit its vulnerability. These hosts will include SRCs and other sites needing to access SDP Science Data Products. We also assume that the SRCs will complete appropriate security audits...”

“We should offer some protection against non-malicious attacks, such as software bugs or human error, though currently this view (AD01) focuses on external attacks.”

Like performance, security is not something you can add on after a design is otherwise complete. Good architectural decisions cannot guarantee good performance or security. Bad architectural decisions can, however, guarantee bad performance or security. Performance and security are not boxes you can bolt onto the side of an architecture diagram. They are pervasive aspects of a design.

5. Executive Summary

The Science Data Processor component of the Square Kilometre Array will exist to provide a service. We wish the system to continue to provide the service, to remain accessible to those it serves, and to use resources to service its design functions, not the ends of attackers. That is to say, we need the system to be reasonably *secure*, but the security is a means to the end of providing the SKA Observatory with a service, not an end in itself.

This document covers different areas of security potentially applicable to the SKA and the SDP. It begins mentioning security mechanisms that overcome the constraints of HPC environments such as the Science DMZ security framework (RD01) and introduces the concept of containerisation (RD03) and reduction of complexity by proposing a minimalist OS for the SDP (RD02).

We submit that there is a credible security risk to the SKA, possible with threats similar to SCADA systems. We recommend (section 8.8) that since vulnerabilities overlap so much with bugs, all software developed for the SKA should be checked using inspections and static checking programs, that the cost of using Linux features such as stack protector and heap protector should be experimentally determined for some of the SKA software in order to set policies for using them and that at least two static checking tools for each programming language to be used in the SKA should be evaluated on existing code to determine their suitability. Because...less insecure code is less buggy code!

The memo also covers modern Hardware/OS related security solutions such as HSM, KMS, vTPM, Secure Boot and looks at low-level vulnerabilities recently released (August 2018) like Foreshadow/L1TF. If we have to design a Security Strategy for a “simple cluster” connected to the internet, in a HPC environment and avoid degrading performance (due to Meltdown/Spectre), we propose integrating OpenStack with Kubernetes (as CERN is doing) and improving security through improving and automating software delivery practices.

It is virtually certain that the SDP *will* be attacked. It is extremely likely that at least one attack during its lifetime will be successful. To put it another way, in this as in other aspects we must *expect failure* and prepare to deal with it.

Summary

- Have a strongly modular design
- With explicit, narrow, **enforced** interfaces.

- Use multiple usernames, most internal, with strong authentication, and much use of UNIX file permissions to limit privileges.
- Have a single shared logging facility but different programs having their own logs.
- Limit the environment and restrict changes to it.
- Refer to SABSA methodology to implement controls in proportion to risk of loss.

6. References

Applicable Documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

Reference Number	Reference
AD01	SDP System-level Security View. SKA-TEL-SDP-0000013

Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

Reference Number	Reference
RD01	Security in HPC Environments. Sean Peisert. Communications of the ACM - September 2017 - Vol. 60 - No.9
RD02	SDP Memo 063 - Considerations for the SDP Operating System. Nicolas Erdödy. August 2018.
RD03	SDP Memo 051 - Cloud Native Applications on the SDP Architecture. Piers Harding. August 2018.
RD04	Software-defined networks in large-scale radio telescopes. 2017. Broekema, Twelker, Romão, Grosso, Nieuwpoort, & Bal
RD05	vTPM: Virtualizing the Trusted Platform Module. 2006. Berger, Caceres, Goldman, Perez, Sailer, van Doorn. IBM T.J. Watson Research Center. http://domino.research.ibm.com/library/cyberdig.nsf/papers/A0163FFF5B1A61FE85257178004EEE39/\$File/rc23879.pdf
RD06	UEFI Secure Boot in Modern Computer Security Solutions. 2013. Richard Wilkins, Brian Richardson. https://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf
RD07	SCONE: Secure Linux Containers with Intel SGX. November 2016. David Eyers et al. 12th USENIX Symposium on OS Design and Implementation. https://www.usenix.org/system/files/conference/osdi16/osdi16-arnautov.pdf

RD08	TaLoS: Secure and Transparent TLS Termination inside SGX Enclaves. 2017. David Evers et al. https://www.doc.ic.ac.uk/research/technicalreports/2017/DTRS17-5.pdf
RD09	CERN: Computer & Grid Security. 2012. Stefan Lüders. ITU Tutorial, Geneva. https://www.itu.int/en/ITU-T/studygroups/com17/Documents/tutorials/2012/11-CERNComputerandGridSecurityITU(2012).pdf

7. Four Aspects of HPC Security

Historically security for HPC systems has not necessarily been treated as distinct from general-purpose computing, except, typically, making sure that security does not get in the way of performance or usability. This section (following RD01) will introduce four key themes highlighting HPC distinctiveness and how they could apply to SKA/SDP.

7.1. High-performance by definition

HPC systems are high-performance by definition, typically performing some set of mathematical operations for the purpose of modeling and simulation, and data analysis. There is a reluctance to take on a solution that might impose overhead on the system because of perceived as a waste of cycles and unacceptable delay of scientific results. This view could frame the types of security solutions that historically might have been considered acceptable to use.

7.2. Distinctive types of operation

HPC systems tend to have distinct modes of operation i.e. compute nodes may be accessed exclusively through some kind of scheduling system on a login node, from which the computation is submitted to the scheduler. And it may be the case that a narrow range of programs exist compared to those commonly found on general-use computing systems. In (RD02) we argue that a “Minimalist OS” would be enough to deal with a specific system such as the SDP.

7.3. Custom operating system stacks

HPC systems often have highly custom stacks, for example:

- Cori Phase 1 at NERSC runs a full SUSE Linux distribution in the login nodes and Compute Node Linux (CNL), a lightweight version of the Linux kernel.
- Mira at Argonne Labs run Compute Node Kernel (CNK), a Linux-like OS for compute nodes, but support neither multi-tasking or VM.
- Aurora, scheduled to be installed in Argonne in 2019 will run Cray Linux -a full Linux stack on its login nodes and I/O nodes, and mOS (*) on its compute nodes.
- Summit, recently installed at Oak Ridge is based on both IBM POWER9 CPUs and NVIDIA Volta GPUs, is running Red Hat Enterprise Linux (RHEL) version 7.4. Summit nodes are connected to a dual-rail EDR InfiniBand network providing a node injection bandwidth of 23 GB/s. Nodes are interconnected in a Non-blocking Fat Tree topology.

(*) [mOS for HPC](#) is an OS research project at Intel, targeting extreme scale HPC systems. It aims to deliver a HPC environment with the scalability, low noise, and repeatability expected of lightweight kernels (LWK), while maintaining overall Linux compatibility that HPC applications need. mOS for HPC remains under development at this time.

So we see that:

- Full OSs are used for login nodes
- Sometimes full OS or lightweight Linux API-compatible version are used for compute nodes
- Bespoke OS exist for single-user only with no virtual memory capabilities or multitasking

At least for the full OS, it is reasonable to assume that the OS contain similar or identical capabilities and bugs as standard desktop and server versions of Linux, and are just as vulnerable to attack via various pieces of software (libraries, runtime and application) that are running the system.

7.4. Openness

Many of these HPC systems are devised in a way that scientists from all over the world whose identities have never been validated may use them. They do not have traditional firewalls (*) between the data transfer nodes and the internet, or the ability to ensure that no physical connection to the regular internet is possible. While it is very positive for science, it is a challenge for security as we will see in the next chapter regarding the SKA.

() Any system that enables SSH from remote is open to a routing exploit*

7.5. Summary

RD01 provides a comprehensive analysis of security mechanisms and solutions that overcome the constraints of HPC environments, such as the Science DMZ security framework which defines a set of policies to address the needs of scientific environments with high network throughput needs.

Other solutions that can leverage HPC distinctiveness as a strength would be analysing system behaviour with machine learning and collecting better audit and provenance data: What are people running on HPC systems?

Looking to the future, we are starting to see an increasing shift toward the use of new virtualised environments for additional flexibility. The notion of “containerisation” may well be a key benefit to security because it simplifies the operation of the machine, and the reduction of complexity is also often a key benefit to system robustness, including security. RD02 and RD03 provide extensive information respectively on containerisation and reduction of complexity by proposing a minimalist OS for the SDP.

8. Security and the SKA: Why it needs to be considered

Wikipedia defines Computer Security as “the protection of computer systems from the theft and damage to their hardware, software or information, as well as from disruption or misdirection of the services they provide”.

“Computer Security includes controlling physical access to the hardware, as well as protecting against harm that may come via network access, data and code injection”.

This section will look at several security problems that would have to be addressed for the SKA.

8.1. Lessons learned from industry - Triton

The 10 February 2018 [issue](#) of comp.risks mentions the Triton attack on an industrial system in the Middle East (“[Menacing malware shows the dangers of industrial system sabotage](#)”). According to [this article](#) it exploited a zero-day flaw in devices used to bring an industrial process to a safe state. Triton “has the capability to scan and map the industrial control system to provide reconnaissance and issue commands to Tricon controllers. Once deployed, this type of malware, known as a Remotely Accessible Trojan (RAT), controls a system via a remote network connection as if by physical access”.

We don’t want that to happen to the SKA.

8.2. Aren’t we lucky to use Linux!

In the same issue of comp.risks, we read that “three NSA exploits previously leaked by The Shadow Brokers have been tweaked so they now work on all vulnerable Windows 2000 through Server 2016 targets, as well as standard and workstation counterparts.” And yes, this means Windows 10 too.

Remote command and control injected through an SMB weakness.

Aren’t we lucky to use Linux!

No.

<https://www.samba.org/samba/security/CVE-2017-7494.html> describes a vulnerability in Linux SMB support. CVE-2017-7494 has been in all versions of Samba since 3.5. It was reported in November 2017. It allows “a malicious client to upload a shared library to a writable share, and then cause the server to load and execute it.”

Not the first Linux CVE, nor the last.

We were worried about the Intel Management Engine. This is an autonomous subsystem on recent Pentium chips which can exert complete control over the Pentium. There is nothing that software (BIOS, OS, library, application) running on the Pentium can do to prevent the IME seeing or altering any data. This is deliberate.

You cannot disable the IME completely without completely disabling the Pentium. Which brings us to the next section:

8.3. Meltdown and Spectre

Meltdown is a hardware vulnerability affecting Intel x86 microprocessors, IBM POWER processors, and some ARM-based microprocessors. It allows a rogue process to read all memory, even when it is not authorized to do so. A purely software workaround to Meltdown has been assessed as slowing computers between 5 and 30 percent. (Wikipedia; other sources agree.). Initial patches were buggy.

Spectre is a side channel attack exploiting a flaw in branch prediction hardware on Intel, AMD, ARM, and POWER machines. CVE-2017-5753 (bounds check bypass, spectre-v1) and CVE-2017-5715 (branch target injection, spectre-v2), have been issued. JIT engines used for JavaScript were found vulnerable. A website can read data stored in the browser for another website, or the browser's memory itself.

“As Spectre is not easy to fix, it will haunt us for quite some time.” (Refer to section 9.5). It’s a class of problems; the two CVEs so far won’t be the last of their kind. Recent patches slow machines down and cause instabilities. Spectre can allow malicious programs to induce a hypervisor to transmit the data to a guest system running on top of it.

8.4. But who would attack the SKA?

The Search for Extraterrestrial Intelligence BOINC system (Berkeley Open Infrastructure for Network Computing) succumbed to a distributed denial of service attack in July 2004.

8.4.1. Attacks and accidents

- The Search for Extraterrestrial Intelligence BOINC system (Berkeley Open Infrastructure for Network Computing) succumbed to a distributed denial of service attack in July 2004.
- “A misconfiguration or failure of a sensor may cause the network to be flooded by packets forwarded on all ports. In effect this failure mode may be considered a self-inflicted DDoS attack.” (RD04 - Broekema et al., “Software-defined networks in large-scale radio telescopes”).
- That is, “attacks” and “accidents” can be similar.
- The Zadko telescope near Perth was cyber-attacked in August 2017.
- Just when astronomers wanted to observe a rare and important event, they were unable to steer their telescope for two days.
- Why would their attacker **not** attack the SKA?

8.4.2. Thefts of physical components

- In December 2017, China opened a 0.6 mile stretch of “solar road”. Five days later it was closed because thieves had stolen one of the panels and damaged others. ([article](#))
- Computing equipment is often stolen.
- Just the network gear alone will be a tempting target.
- “Computer systems are good targets for vandalism.” ([article](#))

8.4.3. Thefts of resources

- On the 12th of February 2018, Australian government web sites (amongst others) were hit by a crypto-jacking attack. The UK too.
- Crypto-jacking is where you take over someone else’s compute to “mine” some cryptocurrency. You don’t care what the machine would otherwise have been doing.
- In December 2017 the Guardian reported that nearly 1 billion visitors to the video sites Openload, Streamango and OnlineVideoConverter were also being crypto-jacked.
- There are other more “traditional” reasons to steal resources: spamming, password cracking, espionage (decryption) amongst others.

8.4.4. Other motivations

- Ransom: “pay us \$\$\$ or we shall destroy your data.”
- Political: “look how incompetent those people are, unable to protect their system”
- Ego: “see what I can do!”
- Accident: you can lose data due to bugs in other people’s code.

8.5. Important issues

- Software security vulnerabilities are often due to errors.
- See the Common Weakness Enumeration at <https://cwe.mitre.org/>
- This means that static checking tools (and dynamic mitigations) for security weaknesses are likely to improve software quality in other ways.
- It’s not a zero-sum game!
- Encryption on all external control or sensitive traffic is a good idea. Authentication of data would likely be sufficient for eg image cubes being sent to the SRC.
- Modern CPUs (SPARC T3 and later, ARM, Pentium, POWER 7+ and later, z9 and later) have hardware support for AES (Advanced Encryption Standard).
- Recent Pentium and AMD systems have a CLMUL (Carry-less Multiplication) instruction to accelerate other cryptosystems.
- Linux supports access-control-list in addition to traditional Unix file permissions.
- It’s already there; let’s use it.

8.5.1. User-space hardening

- Stack protector (B6700 did this in 70s)
- Heap protector (B6700 did this in 70s)
- Address Space Layout Randomisation (B6700 did something similar in 70s)

- Built as Position Independent Executable (ditto)
- Non-executable Memory (ditto)
- *All these help to detect programming errors as well as making attacks harder.*
- **It is not a zero-sum game.**

8.5.2. Be wary of new code

- You can compute cryptographic hashes of programs you intend to run, and refuse to run them if the hashes have changed.
- This detects (some) file system errors as well as providing protection.
- Dynamically executing shell/Perl/Python/Guile &c code is a serious risk. There is a good reason why JavaScript doesn't handle JSON by executing it as JavaScript any more!

8.6. OpenStack

- The OpenStack project includes security work. See <https://wiki.openstack.org/wiki/Security>
- They provide a program, Bandit, to seek security flaws in Python code (like DALiuGE).
- They provide a "lightweight public key infrastructure", Anchor, for short-term certificates.
- They have a list of 139 "security advisories" and 82 "security notes" since 2011.
- There is a book advising developers how to avoid security problems
<https://docs.openstack.org/security-guide/>

8.7. Findings

1. **There is a credible risk to the SKA.**
2. Other Supervisory Control And Data Acquisition (SCADA) systems face similar threats.
3. Techniques devised to protect SCADA and Cloud systems are likely to suit the SKA.
4. Linux supports a range of protection methods with varying scopes and costs.
5. There's stuff worth adopting from OpenStack.

8.8. Recommendations

- Since vulnerabilities overlap so much with bugs, all software developed for the SKA should be checked using inspections and static checking programs, such as PyLint, Flake8, PEP8, NumPy and MyPy for Python. Suitable tools also exist for C, C++, and Fortran.
- Try Bandit and PyLint on DALiuGE (MyPy needs type annotations that the DALiuGE authors should provide).
- Try the language-specific tools on program code (i.e. Fortran tools on the Fortran codes).
- The cost of using Linux features such as stack protector and heap protector should be experimentally determined for some of the SKA software in order to set policies for using them.
- At least two static checking tools for each programming language to be used in the SKA should be evaluated on existing code to determine their suitability.

8.8.1. Draft a “less-insecure-programming” booklet

A short “guide to developing less insecure software” should be produced and given to people writing software for the SKA.

It should cover:

- Using static checking facilities standardly available for C, C++, Fortran, and Python.
- Using additional static checking tools.
- How to do informal inspections.
- How to use valgrind (system for debugging and profiling Linux programs).
- Selected material from OpenStack and CWE.

8.9. Conclusions

- Once again, this is not a zero-sum game.
- **Less insecure code is less buggy code.**

9. Hardware/OS Related Security

This section collates state-of-the-art Hardware/OS related security solutions. We believe that will help the SDP for security test point. Also could be read in conjunction with (RD02) and (RD03) -“Security at OS level in next generation HPC systems: possible applications to the SDP”.

9.1. Hardware Security Module

A Hardware Security Module (HSM) is “a physical computing device that safeguards and manages [digital keys](#) for [strong authentication](#) and provides [cryptoprocessing](#).” (Wikipedia).

In August 2018, Google launched “[Cloud HSM](#)” in beta release -a cloud-hosted Hardware Security Module service that allows to host encryption keys and perform cryptographic operations in a cluster of [FIPS 140-2 Level 3](#) certified HSMs. It is a “[fully managed service](#), (to) protect your most sensitive workloads without needing to worry about the operational overhead of managing an HSM cluster”.

9.2. Cloud Key Management

How do we know if the OS/libraries that we are going to use in the SDP/SKA are trustworthy? We need keys to be sure about that.

Google offers a Cloud Key Management Service ([KMS](#)) that allows you to keep encryption keys in one central cloud service, for direct use by other cloud resources and applications, to always be the custodian of your data, managing encryption in the cloud the same way as you do on-premises, having a provable and monitorable root of trust over your data.

9.3. Virtual Trusted Platform Module

Trusted Platform Module (TPM) is an international standard for a secure cryptoprocessor, a dedicated microcontroller designed to secure hardware through integrated cryptographic keys.

In 2006 IBM presented vTPM: Virtualising the Trusted Platform Module -the “design and implementation of a system that enables trusted computing for an unlimited number of virtual machines on a single hardware platform.”

“As a result, the TPM’s secure storage and cryptographic functions are available to OS and applications running in virtual machines...supports higher-level services for establishing trust in virtualised environments, for example remote attestation of software integrity”. (RD05)

9.4. Secure Boot

Secure Boot is one feature of the latest Unified Extensible Firmware Interface (UEFI) 2.3.1 specification ([Errata C](#)). The feature defines an entirely new interface between operating system and firmware/BIOS.

When enabled and fully configured, Secure Boot helps a computer or system resist attacks and infection from malware. Secure Boot detects tampering with boot loaders, key operating system files, and unauthorized option ROMs by validating their digital signatures. Detections are blocked from running before they can attack or infect the system.

In other words, once the firmware is authenticated, UEFI Secure Boot ensures that only verified firmware components and operating system bootloaders with the appropriate digital signatures can execute during the boot process. Each component executed at this stage must be digitally signed and the signature validated against a set of trusted certificates embedded in the firmware itself. Further details can be found either in “HPE Secure Compute Lifecycle” [white paper](#), Open Compute Project “[Security Project](#)”, or UEFI’s full document (RD06).

9.5. Low-level vulnerabilities

In August 2018 more vulnerabilities emerged in Intel processors that can be exploited by malware and malicious virtual machines to potentially steal secret information from computer memory...forcing to choose either VM security or performance; now you can’t have both with the Xeon architecture in some cases.

Three articles discusses them:

- [“Three more data-leaking security holes found in Intel chips](#) as designers swap security for speed: Apps, kernels, virtual machines, SGX, SMM at risk from attack.” (14/08/2018)
- [“Foreshadow/L1TF vulnerability](#) is forcing Intel customers to choose either VM security or performance” (20/08/2018)
- [“Getting to the Root of Security](#) with Trusted Silicon” (22/08/2018)

Are we moving towards the “Google way”? Google-designed hardware, Google-controlled firmware stack, Google-curated OS images, a Google-hardened hypervisor, as well as data center physical security and services. Plus [Google Titan: Security all the way into silicon](#). Is this the “new normal”?

9.6. Containers

If we are looking for a Security Strategy for a “simple cluster” connected to the internet, how do we do it in a HPC environment to avoid degrading performance? (Meltdown/Spectre).

We need a trusted technical architecture – and verify the complete stack. Services on the OS -if compromised, need to be contained.

We understand that the blueprint should converge to SKA's Regional Science Centres, therefore there should be a standardisation on OpenStack -a convergence on software & hardware: Everyone needs to share the same code/standards which would imply lowering entry barriers/down-selecting core components.

Kubernetes is a strong candidate and it is taking over OpenStack ecosystem (RD03) i.e. CERN is [integrating OpenStack with Kubernetes](#) . These links present Kubernetes' [Pod security policy](#) (Pod is a basic scheduling unit in the Kubernetes: A pod consists of one or more containers) and how to [configure a security context](#) for a pod or container.

There are very strong links between the Cloud Native Computing Foundation SDLC model and improving security through improving and automating software delivery practices.

The NZA team have argued that we should provide curated container base images and libraries. These can be enforced by implementing strong Testing Gates that reject any commits that base containerised software on unapproved base images and libs - the administration of this is easily implemented through CI/CD. For example, the prototype presented for ARL tests Python code quality, and runs against the [Clair](#) container vulnerability scanner.

<https://gitlab.com/piersharding/arl-devops/blob/master/.gitlab-ci.yml>

9.7. Intel SGX

Intel Software Guard Extensions ([Intel SGX](#)) are an architecture extension designed to increase the security of application code and data.

SCONE is Secure Linux Containers with Intel SGX. It increases the confidentiality and integrity of containerized services. The secure containers of SCONE feature a trusted computing base (TCB) of only 0.6x–2x the application code size and are compatible with Docker. SCONE does not require changes to applications or the Linux kernel besides static recompilation of the application and the loading of a kernel module (RD07)

TaLoS is a drop-in replacement for existing transport layer security (TLS) libraries that protects itself from a malicious environment by running inside an Intel SGX trusted execution environment. By minimising the amount of enclave transitions and reducing the overhead of the remaining enclave transitions, TaLoS imposes an overhead of no more than 31% (RD08) with the Apache web server and the Squid proxy.

9.8. Keystone

[Keystone](#) is an open-source project for building [trusted execution environments](#) (TEE) with secure hardware enclaves, based on the [RISC-V](#) architecture. A project initiated in 2018 in Berkeley and MIT, it will have full stack implementation. Keystone v0.1 (2018) will run on FireSim (Berkeley) for FPGA.

“Keystone is intended to be a component for building a TEE that's isolated from the main processor to keep sensitive data safe. TEEs have become more important with the rise of public cloud providers and the proliferation of virtual machines and containers. Those running sensitive workloads on other people's hardware would prefer greater assurance that their data can be kept segregated and secure.” (https://www.theregister.co.uk/2018/08/31/keystone_secure_enclave/)

10. Architecture Considerations for Security

The Science Data Processor component of the Square Kilometre Array will exist to provide a service. We wish the system to continue to provide the service, to remain accessible to those it serves, and to use resources to service its design functions, not the ends of attackers. That is to say, we need the system to be reasonably *secure*, but the security is a means to the end of providing the SKA Observatory with a service, not an end in itself.

It is virtually certain that the SDP *will* be attacked. It is extremely likely that at least one attack during its lifetime will be successful. To put it another way, in this as in other aspects we must *expect failure* and prepare to deal with it.

Like performance, security is not something you can add on after a design is otherwise complete.

Good architectural decisions cannot guarantee good performance or security. Bad architectural decisions can, however, guarantee bad performance or security.

Performance and security are not boxes you can bolt onto the side of an architecture diagram. **They are pervasive aspects of a design.**

10.1. Summary

- Have a strongly modular design
- with explicit, narrow, **enforced** interfaces.
- Use multiple usernames, most internal, with strong authentication, and much use of UNIX file permissions to limit privileges.
- Have a single shared logging facility but different programs having their own logs.
- Limit the environment and restrict changes to it.
- Refer to SABSA methodology to implement controls in proportion to risk of loss.

10.2. Details

10.2.1. The Principle of Least Privilege

This is an ideal that can be approached rather than reached.

However, it is just common sense that if one program has to do two tasks, it needs all the privileges required for both of them, but if the work is divided among two programs, each of them needs only the privileges for its particular task. The same applies to users. Access to file system objects (including POSIX shared memory segments in Linux) can be controlled using users, groups, and access control lists, but this relies on there being multiple programs, multiple users, and multiple groups.

This fits well with the traditional UNIX design philosophy, but not with the “one giant application” philosophy. The drive to “separate the rôles” may be the least comfortable aspect of secure architecture for some designers.

10.2.2. Authentication

Verifying that the originator of a command or request to the system is who they say they are.

There are many forms of authentication, from simple password schemes to biometrics and hardware tokens. Some are easy to break, some are hard. Some require more software and management at the client end, some less. Different levels may be appropriate for different roles.

Seven of the top twenty-five weaknesses involve authentication or authorisation. System boundaries need to be very clear.

The architecture must clearly define data and processing components, identify rôles and actions, and assign permissions to data and privileges to processes just adequate for the system to function.

It is very tempting in a system like the SDP to say “well, it's all one application really, so we only need one user.” **But if that user is hacked, everything is accessible.** Ideally, each different resource should be owned by a different user, and requests from other users to modify it should be authenticated.

Since data and work must be able to migrate in order to allow a failed processor load to be taken up elsewhere, this cannot be pushed to an extreme. However, each data file should only be writable by the process(es) explicitly intended to write it and should only be readable by the process(es) explicitly intended to read it. This will normally involve processing stages running with different credentials.

It must be impossible to bypass authentication.

Authentication must not rely on IP addresses because IP addresses can be spoofed and legitimate users can connect from unusual addresses).

A well established authentication system such as Kerberos should be used.

Connections should be authenticated at both ends.

There should be a discussion around authorisation - what software components have permissions to perform what actions. The first step is that they authenticate themselves, but there still needs to be some sort of authorisation service running (either explicitly or implicitly).

10.2.3. Auditing / logging

Adequate logs are important for intrusion detection and incident response.

The issue here is that the logging needed for routine operations and maintenance and the logging needed for intrusion detection and incident response, while overlapping, are not the same. Again we may take DALiuGE as an example: it uses logging for debugging, performance analysis, and error reporting. It does not log all the information one would want for security purposes.

It is important to write *enough* information to logs to allow analysis of faults and attacks, but not so *much* that it is hard to find needles in a cloud of straw or be the vector of a denial of service attack.

Every security-critical event must be logged in sufficient detail. To make it easy for inspection to find places where logging occurs (and by omission, where it should be does not) a single shared logging module should be used. This should not be direct calls to `syslog(3)`, although `syslog(3)` might be the appropriate support.

Such a module can monitor the rate at which the log files are growing so that an attack or fault cannot overwhelm the system.

Data from untrusted sources should not be written to logs without being sanitised. Conversely, sensitive information should not be revealed in log files if they might be accessible. Again in connection with data, information should be recorded in a consistent way, a canonical way if possible. (So file names should be logged using `realpath(3)`.) Again, this argues for a logging module whose interface is sensitive to what *kind* of information is being reported and does not (like `syslog(3)`) blithely accept any old string.

The principle of least privilege applies to log files as much as anything else. A program needs the right to append to its log file(s), nothing else, and in particular, not to other program's log file(s) nor any ability to modify logs. (Some operating systems can grant a program the privilege of appending to a file without granting it the privilege of truncating or overwriting it. UNIX cannot. Ensuring that log access is only through a library module helps.) A log analysis program needs the right to read log files, no more.

All attempts to perform “dangerous” operations should be logged, both successful and unsuccessful.

10.2.4. Encapsulation

If one component is breached, we do not want that opening up the whole system.

This argues for a design that makes much use of separate processes communicating through channels that cannot express bad things. That introduces communication overheads that would not be incurred by components living in the same address space. Languages and operating systems like M# (a C# dialect) and Midori (the green-field operating system M# was designed for) or their Burroughs predecessors can provide logical encapsulation in a shared address space. Languages like unrestricted C, C++, and Fortran cannot.

Three of the top four weaknesses involve trusting a string from an external source: SQL injection, OS command injection, and cross-site scripting. Another in the top 25 is “uncontrolled format string”, and yet another is Improper Limitation of a Pathname to a Restricted Directory, where an unchecked file name may use `..` to point to places it shouldn't. It's not just strings: Reliance on Untrusted Inputs in a Security Decision and Download of Code Without Integrity Check are of the same kind.

Explicit, narrow, **checked** interfaces are the key here. The text for CWE-22 puts it so well there's no point in paraphrasing: “**Assume all input is malicious.** ... Reject any input that does not strictly conform to specifications When performing input validation, consider all potentially relevant

properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules.”

The level of checking here goes beyond what RPC or CORBA can specify.

For example, Google's Protocol Buffers

- do not let you set limits (other than C type) on integers;
- do not let you set limits (other than C type) on floats;
- do not let you set size limits on strings;
- do not let you place content restrictions (such as “is a well formed e-mail address”, “is a well formed host name”, “is a relative file name without . or ..”) on strings; and
- do not let you place bounds on repeated elements.

The same can be said of Thrift. Some thought should be given to extending Protobufs or Thrift, or even creating a new interface language, although “new” often means untested and potentially vulnerable.

10.2.5. Encryption

Successfully decrypting a data stream proves that the sender and receiver have some knowledge in common.

Several modern CPUs have special support for encryption: zSeries, SPARC T4 and T5, ARM8, Power8 and 9, and of course recent Pentiums with the AES instructions. The overhead of hardware tuned encryption/decryption is tolerably low, setting up an encrypted connection is more expensive than setting up an unencrypted one, so using encryption internally suits a design where communication channels are infrequently set up and torn down, being used for relatively long periods of time, and does not suit a design where connections are very frequently set up and torn down.

Encryption is important for blocking eavesdropping and tampering (man-in-the-middle attacks). Communications between the outside world and the SDP should certainly at a minimum be authenticated.

The SDP is a HPC system. Data that is only accessible internal to the system and non-sensitive need not be encrypted, unless this can be done using hardware acceleration so that performance is not sacrificed. We probably are not too worried about data flowing out of the system either, although using cryptographic message digests to ensure that the data a client receives is the data the SDP sent would be a good idea. The key worry is *commands* coming in. The current standard for encrypted network access is TLS. This requires great care with certificates.

We should be specific about what communications need encryption - encrypting visibility data would be very expensive due to the data rates. Even calculating a hash code would involve a small compute load, albeit much less than encryption of all the data.

If using AES as suggested, it'll require session key negotiation between the parties, so there will need to be some PKI in place, probably between the software components communicating. Generally the session keys get periodically renegotiated. Refer to 9.2 for Key Management.

10.2.6. Hardening

Attacks often depend on software errors, and these errors often fall into well known categories that verification tools can catch.

Hardening here refers to adopting coding and checking practices that catch such errors before they can be exploited. The book "Exploiting Software: How to Break Code" has hundreds of pages describing common weaknesses. Hardening against such weaknesses suggests selecting languages for which suitable checking tools are available. It also favours designs with typed interfaces (like Ada packages, RPC IDL, CORBA IDL, Protobufs, Thrift, or Joe Armstrong's UBF).

Here are some of the top 25 weaknesses:

- CWE-120 Buffer Copy without Checking Size of Input
- CWE-131 Incorrect Calculation of Buffer Size
- CWE-190 Integer Overflow or Wraparound

In a personal note, one of us did undergraduate computing on a Burroughs B6700, where array bounds checks **couldn't** be bypassed (they were part of the hardware instructions for array accessing, and you just could not access array elements any other way) and where integer overflow was a hardware-reported runtime exception (that again, **couldn't** be suppressed). Imagine the shock when discovering that other machines did not do this.

In a language like Lisp (1960) or Smalltalk (1980) there is no such thing as integer overflow (why should hardware register sizes surface in programming languages?) and array bounds are always checked. While we are wary of some aspects in DALiuGE, its use of Python cannot be faulted here: Python is not subject to these weaknesses. (Of course they are subject to weaknesses of their own.) More promisingly, we can think of the SDP as a giant embedded system, and there is a programming language specifically designed for writing embedded systems that are both reliable and efficient. That is Ada -probably not the language of choice for the SDP. Ada provides checked arrays and check integer arithmetic, with good Ada compilers going to some trouble to optimise away unnecessary checks. Maybe -for practical reasons, the SDP is going to be programmed in some mix of C, C++, and Fortran (probably using OpenMP) with scripting in Python.

For C, C++, and Fortran, it's worth noting that GCC [offers](#) `-fsanitize=signed-integer-overflow`, `-fsanitize=bounds`, and a host of other such options. This isn't an architectural issue *per se*, but it does mean that these languages can be used much more safely. There is a run-time cost to all this, of course, but there should be a policy that all code adopted into the system must be tested this way.

The Java platform provides a security architecture which is designed to allow the user to run untrusted bytecode in a "sandboxed" manner to protect against malicious or poorly written software. This "sandboxing" feature is intended to protect the user by restricting access to certain platform features and APIs which could be exploited by malware, such as accessing the local filesystem, running arbitrary commands, or accessing communication networks. As part of Oracle's Fusion Middleware Security Guide, Oracle provides an [Overview of Java Security Models](#).

[Critics](#) have suggested that updated versions of Java are not used because there is a lack of awareness by many users that Java is installed, there is a general lack of knowledgeability on how to

update Java, and (on corporate computers) many companies and organisations restrict software installation and are slow to deploy updates.

One of the prevention techniques for CWE-190 is **Ensure that all protocols are strictly defined, such that all out-of-bounds behavior can be identified simply, and require strict conformance to the protocol.** This reinforces the need for clearly defined *enforced* interfaces between components.

A useful property of Ada is the SPARK subset. AdaCore provide a verifier for SPARK. They like to boast that in a mission that launched twelve CubeSats designed by students at various universities, the one and only CubeSat that completed its mission was programmed in SPARK Ada and verified. Most of the others failed due to software errors. Of relevance to the SDP: there is a similar tool for C, called [Frama-C](#). That's a .com site, and there are commercial licences with support, but it's also available under the LGPL. MathWorks have a commercial product called Polyspace, which can do impressive things for C and C++, but do not care to provide prices on the web.

Separation of functions into separate UNIX processes communicating through pipes and files is itself a form of hardening. If you avoid shared memory (such as memory-mapped files) it is impossible for one process to tamper with another's memory.

10.2.7. Tainting

The basic idea of tainting is to distinguish untrusted (tainted) data from trusted (clean) data and to limit the things that can be done with unchecked tainted data. Perl was the first scripting language to offer taint mode. There are versions of Ruby, PHP, and Python that do this. There are static checking tools to do taint checking for statically typed languages. The key issue for architecture here is to keep tainted and clean data separate and to be clear about what "sanitation" is needed. DALiuGE, for example, does not mention 'taint' anywhere in its *.rst documentation or its *.py source code. It does describe validation of logical graphs, but communication between internal nodes is totally trusting. For example, one RPC scheme it supports is ZeroRPC, and the PyCon 2012 presentation on ZeroRPC says flatly, "Security: there is none."

10.2.8. Control the environment

This is basically [mitigation 3](#) from the CWE.

Run as much code as you can as ordinary users. Run under virtualisation with a minimal set of libraries and tools. Lock down the libraries: do not allow new code to be installed. In particular, there should be no dynamic loading of kernel modules. Device drivers should be statically linked to the kernel. Make as many files as possible read-only and use file system protections to limit write access to the rest. Access to features such as `sysctl(2)` and `/proc/sys` should be very limited. SNMP is useful for monitoring and managing distributed systems, but parameter changes through SNMP need to be authenticated.

This document was written with constant reference to the Common Weakness Evaluation (CWE).

The [Common Vulnerability and Exposures](#) "is a list of entries—each containing an identification number, a description, and at least one public reference—for publicly known cybersecurity

vulnerabilities.” In April 2018, it contained **more than 100,000 vulnerabilities**. System administrators need to make sure they have installed patches for the vulnerabilities in their systems.

System architects and programmers need something else: they need to know how to *avoid* creating such vulnerabilities in the first place. That is the purpose of the [Common Weakness Enumeration](#): “A community-developed list of software weakness types”. The current version (3.1) lists 716 weaknesses. Reading it is like taking a course in secure coding.

10.2.9. Proportionality

Of particular benefit would be to include Proportionality of investment and control: implement controls in proportion to **risk of loss**, following the [SABSA methodology](#). “The primary characteristic of the SABSA model is that everything must be derived from an analysis of the business requirements for security, especially those in which security has an enabling function through which new business opportunities can be developed and exploited” ([Wikipedia](#)).

10.2.10. CIA - Key Concepts

Finally a word about the Key Concepts of Information Security - CIA (Confidentiality, Integrity and Availability -plus Non-repudiation) i.e. Availability is an aspect of security which is often overlooked (security is not much use if eg the authentication service is DOSed). Refer to Wikipedia for general information https://en.wikipedia.org/wiki/Information_security#Key_concepts

11. Conclusion

This document presents a mix of generic level advice and specific low-level comments.

Decisions about security need to be aligned with a risk analysis of the SDP, which sets the scope, costs, attacks and benefits of the overall system. Without that it's impossible to evaluate the cost of encryption. The risk analysis should help inform which aspects of the SDP need addressing, and then we can choose the responses from the options that this document is discussing.

For example, if you want to accept code from third-parties, it's a high cost to require them to use a specific programming language eg SPARK Ada, and higher again to require them to have formally proven that their code does what is intended. If you can't bear those costs, you might end up letting them run arbitrary Python code, but then you have to manage the additional libraries, etc., which leads to other costs in other places.

12. Appendix - Debate

A preliminary version of some contents of this memo and of (RD02) were posted in Confluence (<https://confluence.ska-sdp.org/display/WBS/Security+at+OS+level+-applications+to+SDP>) in January 2018 -at the time when the first vulnerabilities (Meltdown and Spectre) were released. This generated a very interesting debate at that time (comments are on that context) within the SDP community -with more than 20 comments.

This Appendix presents an abridged version of some of them -aimed to capture common knowledge toward further work. Note that [Comment] and [Question] are SDP community contributions while [Response] is from the authors. We added subsections just to organise the information flow.

12.1. Security Policy for the SDP

[Question] "...this page assumes that we need a security policy within SDP. Do we? SDP is not a general purpose cluster, but rather a tightly controlled appliance with very few, if any, physical users. We may get away with just securing "at the door". The recommendations above seem overly restrictive considering the way that SDP is intended to work."

[Response] To say that the SKA is "tightly controlled" can only mean that it *has* a security policy and mechanisms to enforce it. Without such policy and mechanisms, there is no such thing as control.

I am reminded of a conversation I had with a student many years ago. He had just installed a server in his student hall of residence for the other students there to use. There were six thousand attempted break-ins within the first 10 minutes after he connected it to the net.

The fact that there will be few users physically present is itself one of those things that makes good security important: if you make it easy for legitimate users to access and control the system remotely, you thereby make it easy for attackers to access and control the system remotely, unless you take steps to ensure otherwise. (This memo cannot fully) discuss these issues at greater length, however there are a few fairly obvious points:

(A) Good security measures **protect against ACCIDENT as well as attack**. And accidents are common. To pick an example at random, comp.risks volume 30 issue 50 has an article <http://catless.ncl.ac.uk/Risks/30/50#subj5> which informs us that a newbie developer "accidentally" tripped over [a previously unknown security] vulnerability, and erased other people's wallets' to the tune of USD 300 million in a cryptocurrency.

Another example was the 2018 Hawaii false ballistic missile alert where human error -or misinterpretation, triggered a massive evacuation which caused panic and disruption throughout the state. Subsequent investigation faulted "insufficient management controls, poor computer software design, and human factors" for the incident. Officials did not name the employee responsible for the error. (https://en.wikipedia.org/wiki/2018_Hawaii_false_missile_alert).

(B) **If the SKA is accessible through the internet, it WILL be attacked.** If it is accessible only through a special network with its own protocols (which might not be a bad idea), but the computers it talks to are accessible through the internet, THEY will be attacked. This raises an important point, that security for remote users needs to be part of the eventual complete plan.

(C) The purpose of the present task is not to recommend a policy but to discuss mechanisms and what they can and cannot achieve, so that an informed judgement can be made later about what policies and mechanisms are acceptable, **considering the price of adopting them and the risks of not adopting them.** (NB: refer to section 10.2.9. Proportionality)

(D) The point of paragraph (B) was that the SKA will be attacked **just because it is there.** EVERYTHING on the internet is probed for vulnerabilities. However, the SKA will be a very tempting target. Sure, the information on it is probably not going to be tempting to thieves. But the SDP itself, as a computing engine, will be tempting.

People with millions of passwords to crack, for example, definitely have a use for it. And since the project is using commercial CPUs and GPUs and a minimally altered and well known operating system, to reduce costs, **this also reduces costs for attackers.** Indeed, if attackers stole the computing resources of 1% of the machines in the SDP, it might be quite a while before anybody noticed. Again, the SDP will have a lot to communicate to the outside world, if you are in the botnet/spamming business, wouldn't you be tempted to steal some of that bandwidth?

We come back to "tightly controlled". Let us hope that the system is tightly enough controlled to stop the resource thieves and botnetters. But the degree and kind of control we wish to exercise is amongst other things a security policy that the mechanisms used to enforce it are amongst other things security mechanisms.

12.2. Why would someone attack the SDP?

[Response] A real world example -where attackers had no apparent gain, is the recent case (October 2017) where "Astrophysicists at Western Australia's Zadko telescope had just learned about the detection of a monumental deep space event involving two neutron stars colliding — which they had been hoping to find for years — when they came under sustained cyber attack."

"At the critical and fleeting moment, they could not move their telescope to track the gigantic explosion 130 million light years away." "Technical experts ... worked non-stop to get the computers back online and protect them from the cyber attack." "But by the time they had... two whole days had passed since the neutron star merger had first been detected."

<http://www.abc.net.au/news/2017-10-17/cyber-attack-almost-costs-team-look-at-colliding-neutron-stars/9055816>

[Comment] "...I have escalated this to the attention of the Risk Review Board as I think you have enucleated in some way all the characteristic of the Forbidden Fruit and I believe this deserves more attention. Personally, I have even added that "It is not difficult to imagine religious sects, hacker

groups or simply idiots trying to drive cyber attacks to have their say about the origin of the Universe (some risks in the long term could even come from official Religions, exactly as Galileo experienced!)"

12.3. Is security a matter for SDP or SKAO?

[Comment] "It was asked (during an SDP MT meeting) if whether we need to do this investigation. Do we actually want to apply sky securities. No need to do work on this. The issue is the architectural one. Architecturally we are isolated. And that's our understanding and security is a system level issue but not an SDP issue. It is clearly the SKAO's ... the issue raised about the non-patched version of the system is the important one and think about the module view SDP has. ...(it's specifically an) operating interface. Another point brought about was that mainly the SKAO would like to impose operations standard across the system."

"Before this meeting, (it has been asked) if a specific risk should be entered in our Risk Register to trigger more work, (and) have been provided with the following statement: "**The SKA architecture means that SDP sits behind TM effectively from a security point of view.**" After the meeting, I shared [Big Data Analytics in Cybersecurity](#) when it mentions the challenges for SKA (10.4.1 Single Big Dataset Security Analysis), but I have been reassured that SDP is isolated in the architecture."

[Comment] "So what is the actual decision on this? Wait for SKAO's input, then map to our architecture (security view)?"

[Comment] "It is true that our security policy can likely be more lax on the inside than that of a comparable general purpose cluster. However, that doesn't mean we shouldn't have one, for the reasons Nicolas listed. I would also add that SRCs will actually look more like general purpose cluster (or even cloud-like) environments, so SDP must at minimum be compatible with running under these kinds of more tightly controlled conditions."

"What exactly this means will have to be determined. Most of the listed recommendations seem common sense from a robustness point of view alone. Main exception would be the unqualified protocol audits: I would agree that control-plane protocols (i.e. "coordination" in the C&C) should be audited and might benefit from fairly tight access control. Furthermore, both storage and data queues should offer at least some level of protection against the application asking for data that it's not supposed to access. On the other hand, there is little point in - say - auditing internal data exchange between execution engine slave instances."

"In practice, I don't expect that this will boil down to a lot more than adopting a few best practices from existing HPC installations. We have a security view on the roadmap for the architecture (NB - RD01), which will document these kinds of things. Doesn't look like it would add more complexity, the C&C should already be fairly well-suited for adding security features."

12.4. "Security Wheel" and how CERN fights hackers

[Comment] "One useful way of looking at a robust security architecture is via the "Security Wheel". The perimeter security is at the rim (needed as mentioned above), and the hub around an

application/service has a hardened operating system, configuration and identity management. Then there are the spokes (core security services): Availability, Authorization, Authentication, Auditing, Compliance, Confidentiality, Integrity, Labelling, Logging, Management, PKI, Policy. Without all the components the wheel is broken.”

“One might argue that some security components, such as confidentiality might be less important in the SDP in some cases (eg for visibility data), but I think most of the others still need careful consideration for relevance. Even confidentiality would be critical for any authentication service in the system, which in turn is critical for authorization.”

“It’s always much easier to consider security from earlier in the design using standard security patterns, and discard any (if any) unnecessary components, than to try retrofitting it later. It need not and should not be too restrictive nor incur a lot of system overhead.”

[Response] [This article](#) “How CERN fights hackers” (2016) presents good insight:

- Security is all about balance—keeping users and data safe (*) has to sit alongside usability and efficiency.
- Security trade-offs: CERN’s priority is balancing secure systems with academic freedom.
- Everyone has to patch and secure their own devices, and their own larger systems too.
- "People are used to having a certain liberty to choose what technology they would like to employ, the hardware they would like to run, the operating system they would like to use, and the applications they would like to install." If not, the vibrancy of CERN's community is under threat. "If we don't do this we will force them into a corner, and all the intelligence, all the creativity will be killed."
- Further detail can be found in “CERN-Computer & Grid Security” (RD09).

(*) NB: we should always assume that the statement “what is safe?” must be considered within a context. More appropriate would be to say “relatively safe -with limits” (Refer to section 10.2.9. Proportionality)

12.5. The impact of Meltdown & Spectre in SDP’s hardware budget

[Comment] “The security discussion is very timely - especially with what is playing out over Speculative Execution exploits ...

<https://www.theregister.co.uk/2018/01/04/intel amd arm cpu vulnerability/> .

... as the details unfold it is becoming increasingly important that the impact of the security implications of this kind of vulnerability is investigated because of the far reaching implications for the SDP.”

- The vulnerability appears to affect CPUs and some GPUs
- The current fixes (which have been cooking for 6 months) are giving a performance hit ranging up to 20%
- These performance impacts are highly variable dependent on workload types, but one that stands out to me is the NvME FIO benchmarking that Redhat has done that indicates 8-19% (<https://access.redhat.com/articles/3307751>). If this translates directly onto the NvMEoF

solution proposals for the SDP then we could have a processing hardware budget impact of >10% along with the implications that this could have for the power cap etc.

- The fixes are systemic starting with kernel patches that enforce separation of user space and kernel memory (no direct access anymore), and will likely evolve into compiler modifications that may or may not create additional overheads.”

“Given the fundamental nature of this problem, all existing processes for quantifying performance, and developing costing models (across the entire industry) are now suspect until we obtain patched software for kernels, libs, programming frameworks, services, and compilers and start re-validating. With this in mind, my recommendation would be to obtain these fixes as soon as possible and start re-running the existing SDP benchmarks (infrastructure, services, application components, the ARL etc.) to understand what the new performance constraints are and their impacts on compute, storage, network - if it's significant it may even impact on architectural choice.”

“Trying to link this back to Security at the OS level (NB -RD02) - there is an ongoing need for threat assessment and platform impact analysis, and these recent events are a timely reminder of the value of the work that you (this memo authors) are doing in relation to this. Picture this - Moore’s Law (on the current technology stack) is in danger of dramatically slowing down, and at the same time the threat levels are going up with the potential to eat away at the expected performance gains - it certainly seems to be putting a squeeze on the growth curve.”

[Comment] “I am not sure the implications for SDP are as far reaching as is suggested.”

- A hardware solution to Meltdown will be implemented before SDP is rolled out.
- The security and operational model of SDP allows us to (potentially) sacrifice security for performance.

“Notwithstanding the above, there is one very important thing we do need to consider: any and all benchmarking we do from this point forward needs to mention whether or not KPTI is enabled and ideally present results with both. Very early indications with I/O dominated applications suggest that the impact may be extreme (i.e. ~40%).”

“There is another consideration that we may have less influence on though. I've heard rumors that Meltdown and Spectre workarounds may be rolled out in BIOS. Whereas KPTI can be turned off at boot time, this may not be the case for a BIOS workaround. A remark about this may need to be added to the procurement documents.”

[Comment] “We can hope that the hardware solution is rolled out before the SDP initial hardware purchase is made - in fact it would need to be before the selection and hardware purchase negotiation is made because anything that impacts on the amount of compute required to provide a predictable platform needs to be done in advance. It is likely (I acknowledge it is early days) that even the hardware solution will need to enforce space separation, or scrub discarded spec-ex space, or remove spec-ex altogether (as in some other architectures). Irrespective of the path taken for fixing this, it seems fair to assume that there will be a cost borne by computation that doesn't exist today (also note that some variants of Spectre have no fix strategy yet).”

“Given that there will be heavy reliance on 3rd party code in the SDP and the small engineering team budget we have, there is no way that every piece of software will be vetted. If we don't sandbox applications from known threats (and this would be sufficiently open knowledge) then the threat level is increased. Additionally, the kind of threat that Meltdown and the Spectre variants are is such that currently they are to all intents and purposes undetectable. If this remains so, then the first time that the exploit is exercised (if even detected?), there will most likely be no choice but to apply the performance penalty. We should be considering contingency for this now, as as stated above, benchmarking on the worst case scenario so that we can some idea of what the penalty is for our workload characteristics.”

“Certainly in the commercial world, the Bastion model of security is untenable. Taking the approach of only securing the perimeter only increases the value of breaking in - once in it is open slather. I think that this also applies to the SDP as prize is resources on scale that currency miners can only dream of - providing the incentive and economic imperative.”

[Comment] “You raise a good point that we don't know what the hardware solution is going to be, or indeed what the performance impact of that solution might be. However, I don't think speculative execution will be removed from future hardware, considering both the fact that only Intel is hit by the more serious vulnerability (Meltdown, thus more secure speculative execution is possible), and the reliance of modern hardware on out-of-order execution for performance.”

“A lot of what we're discussing will need to be system wide policy... While we should not sit back and wait, it does seem like we have time (for SKA) to see what industry comes up with. Tracking progress and considering possible implications as a background task seems sufficient for now.”