Hochschule Konstanz
Fakultät Informatik

Software Security

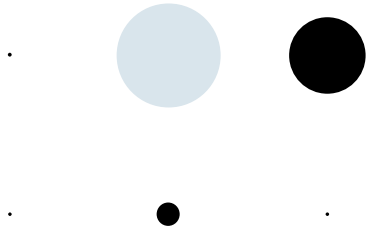
AIN

Hanno Langweg

03 Secure Programming

Secure Programming

- Architectural risk analysis
- Input handling and defensive programming
- List of banned functions
- Data flow analysis
- Handling of user input



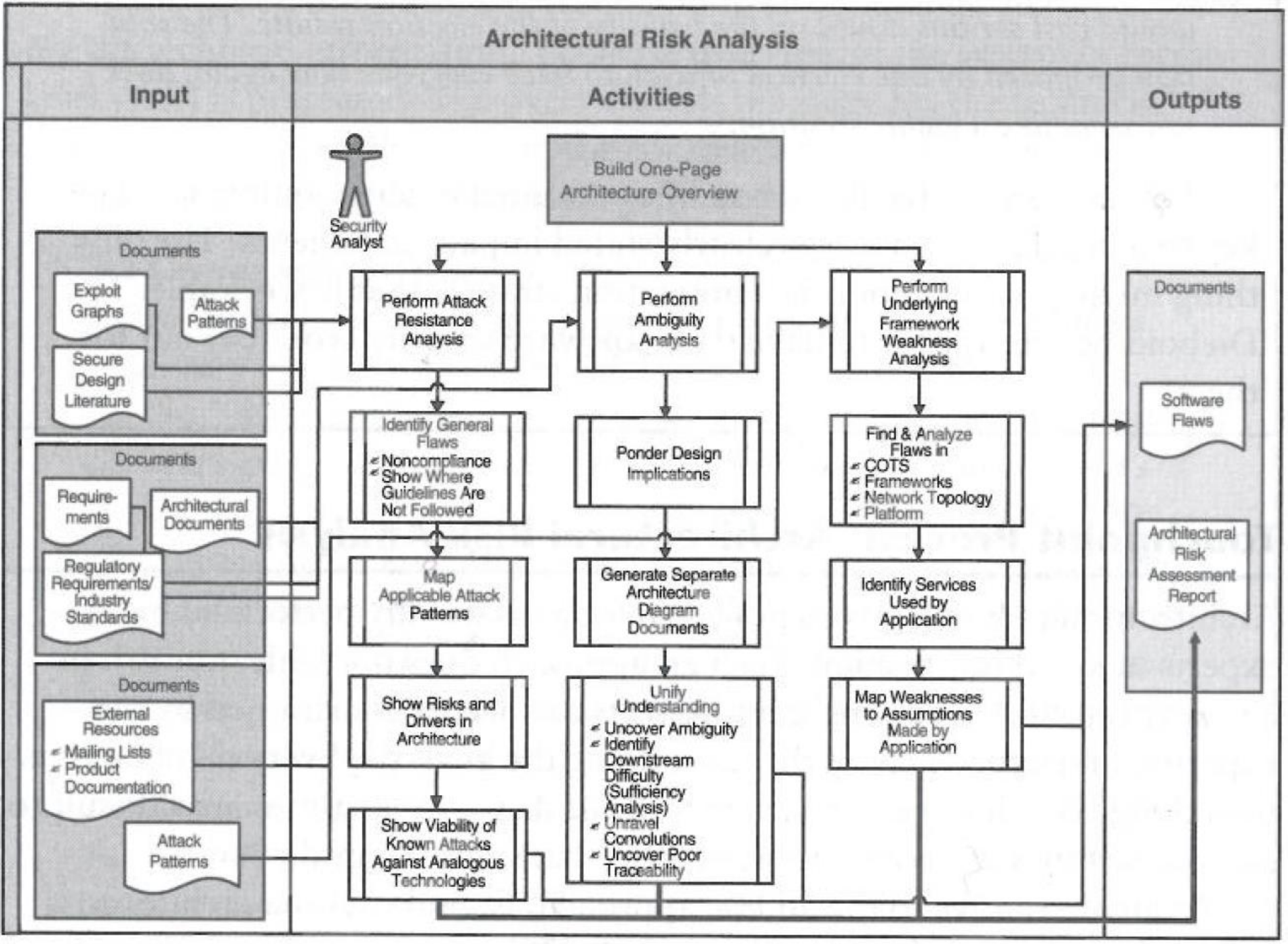
Architectural risk analysis

- Based on high level view
- Known vulnerability classes/attack patterns
- New risks based on misunderstanding
- Underlying platform/framework weaknesses

- Modular architecture simplifies analysis

Architectural risk analysis

- Common Criteria, EAL2-EAL7
  ADV_ARC security architecture description (excerpt)
  - Security features cannot be bypassed.
  - Protection by TOE itself from tampering by untrusted active entities.
  - Description of security domains maintained by the TSF (TOE security functions) consistent with the SFRs (security functional requirements).
  - Secure TSF initialisation process.

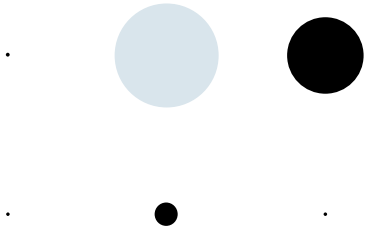

Architectural Risk Analysis
Input
Activities
Outputs
Security Analyst
Build One-Page Architecture Overview
Documents
Exploit Graphs
Attack Patterns
Secure Design Literature
Documents
Require-ments
Architectural Documents
Regulatory Requirements/ Industry Standards
Documents
External Resources
Mailing Lists
Product Documentation
Attack Patterns
Perform Attack Resistance Analysis
Perform Ambiguity Analysis
Perform Underlying Framework Weakness Analysis
Identify General Flaws
Noncompliance
Show Where Guidelines Are Not Followed
Ponder Design Implications
Find & Analyze Flaws in
COTS
Frameworks
Network Topology
Platform
Map Applicable Attack Patterns
Generate Separate Architecture Diagram Documents
Identify Services Used by Application
Show Risks and Drivers in Architecture
Unify Understanding
Uncover Ambiguity
Identify Downstream Difficulty (Sufficiency Analysis)
Unravel Convolutions
Uncover Poor Traceability
Map Weaknesses to Assumptions Made by Application
Show Viability of Known Attacks Against Analogous Technologies
Documents
Software Flaws
Architectural Risk Assessment Report

Known vulnerability classes

- E.g. MS STRIDE threat model
  - **S**poofing of user identity ➔ authentication
  - **T**ampering of data ➔ integrity
  - **R**epudiation of actions ➔ accountability
  - **I**nformation disclosure (e.g. privacy breach) ➔ confidentiality
  - **D**enial of service ➔ availability
  - **E**levation of privilege ➔ authorisation
- https://www.microsoft.com/en-us/sdl/

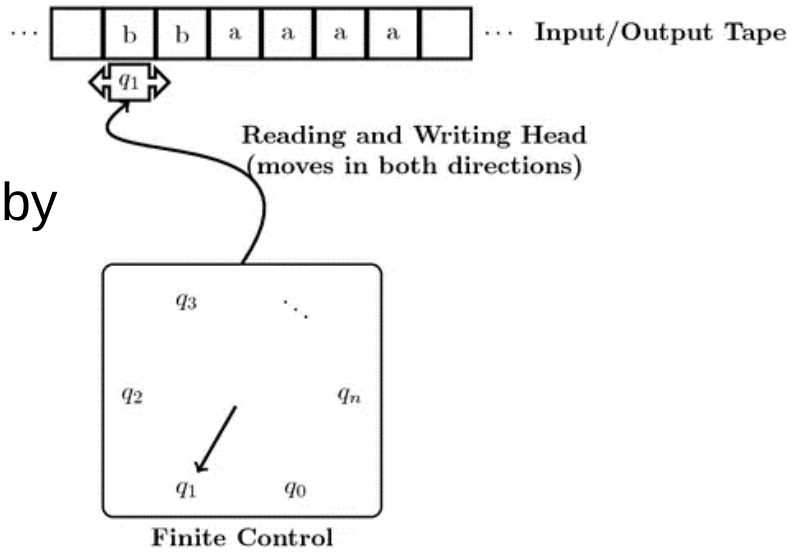
Underlying platform/framework

- Determine impact of external **dependencies**
  - Operating system
  - Deployment/installation/configuration
  - Browser, VM, execution environment
  - Plug-ins, libraries
- Unused product features
- Debug interfaces



Input Processing

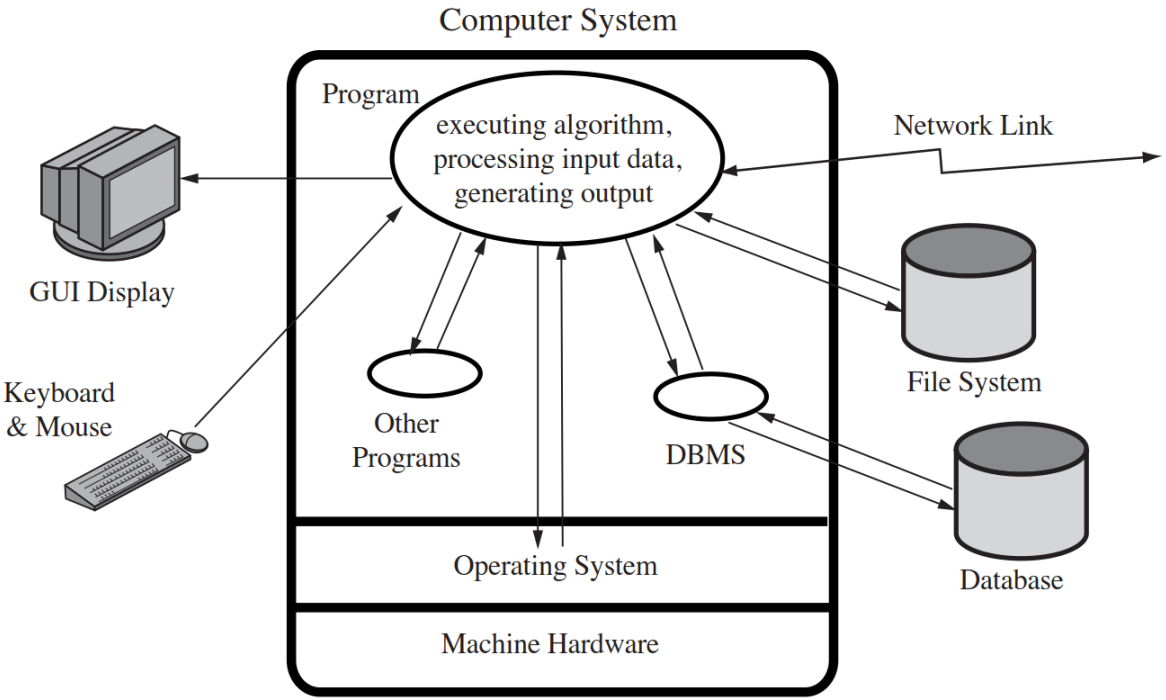
Input processing



- Behaviour of application determined by
  - Code (transition function)
  - Internal state
  - Input
  - User (provides code, input)
- Insufficient checking and validation of input may lead to unexpected control flow

https://i0.wp.com/tuxar.uk/wp-content/uploads/2017/04/turing-machine-2.png

Input sources



Stallings, fig. 11.1

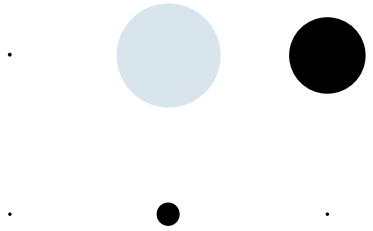
- Files, data bases, USB devices
- Network traffic, web requests
- Sensor data
- System time, performance data
- User interface
- Mobile phone example

Trustworthiness of input

- Who can **create** input?
- Who can **modify** input?
- Who can **select** input?
- How can we **authenticate** input?
- (Who can **observe** input?)

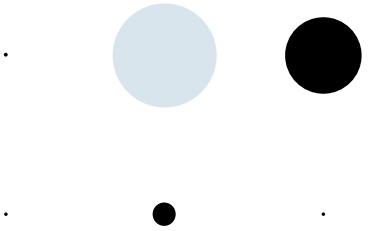
Prevention

- Ensure input can only be created + modified by trusted parties
  - **Authenticate** source
  - **Control access** to source
  - Rely on platform/source to **enforce policy**
- Examples: Access control on files, TLS for network services, restricted access to physical devices



Detection + Reaction

- Ensure unauthorised modification of input can be observed
  - Compare input with baseline
  - Verify properties of input
  - Format
  - Check sums, signatures, access log files
- Correct control flow when unauthorised modification of input is detected
  - Refuse, block, limit access
  - Alert, notify, get confirmation
  - Log, shut down



Input processing

- Web applications
  - SQL injection
  - XSS
  - CSRF
- Input flow tracing (data flow analysis, static analysis)
- C string handling
  - Buffer overflows

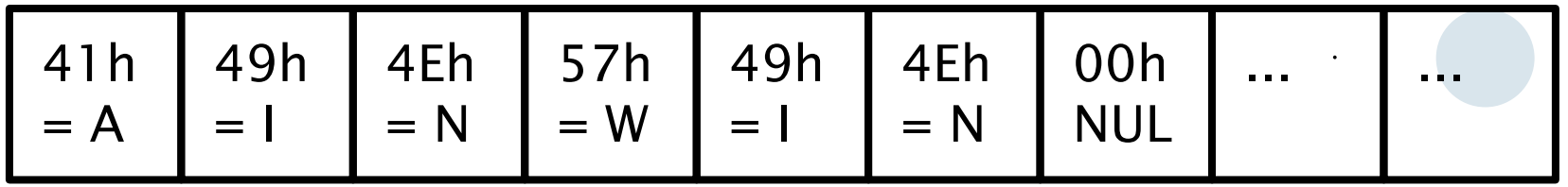
Data flow analysis

- Taint analysis
  - Mark input entry points, follow data through program
  - Explicit passing in arguments, implicit passing in control flow decisions
- Tool-based

- Is untrustworthy input always validated before being used for security-sensitive decisions/operations?
- Can a method be reached by attacker-controlled input?

C string handling

- No native string type ➔ arrays of char
- NUL 0x00 marks end of string,
  chars following NUL not part of string
- Manual management of buffers ➔ error-prone
  - Dynamic allocation (performance, leaks)
  - Who is responsible for buffer management?
    Caller? (What if callee does not respect bounds?)
    Callee? (What if caller does not free memory?)

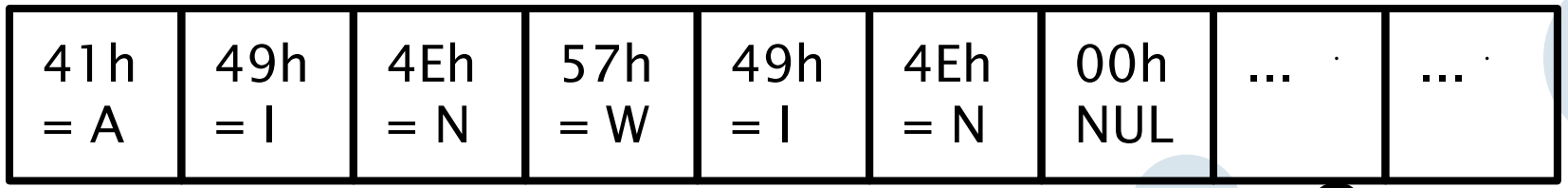


C string handling

- Example scanf()

```
char buffer[7];
buffer[sizeof(buffer)-1] = '\0';
scanf(buffer, "%s");
```

- "AINWIN"

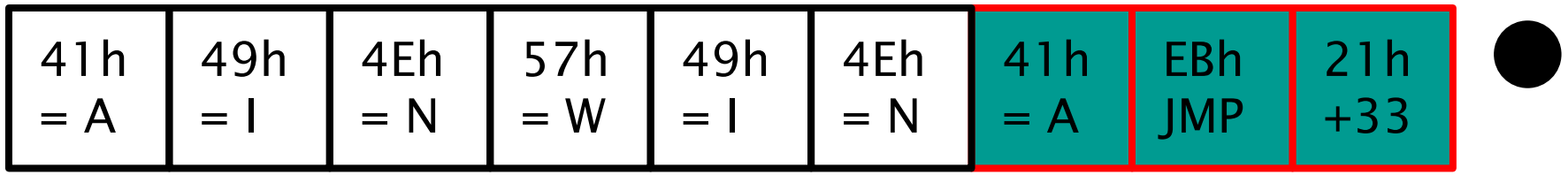


C string handling

- Example scanf()

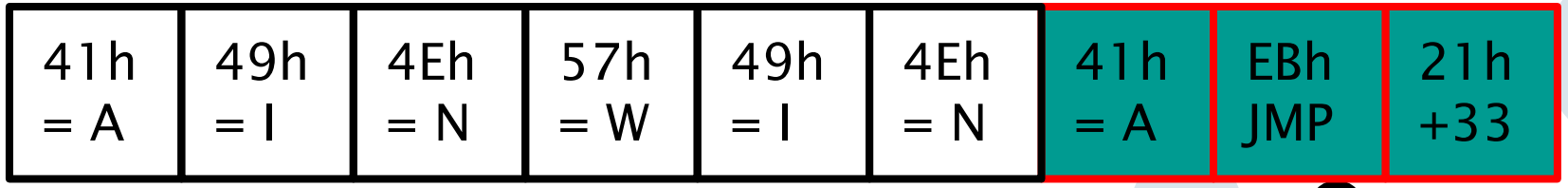
```
char buffer[7];
buffer[sizeof(buffer)-1] = '\0';
scanf(buffer, "%s");
```

- "AINWINAë!"
- Buffer overflow



C string handling

- Two problems introduced
  - Not (or randomly) terminated string
  - Memory overwritten that does not belong to string
- Memory content influences control flow



C string handling

- Classes to abstract away internals
  - C++ string class in standard library
  - String types in all higher order languages
  - Underlying API functions often require C strings
    ➔ conversion, risks in lower layers
- Bounded string functions: Transmit buffer size to callee
  - Size parameter can still be incorrect
  - Microsoft Security Development Lifecycle Banned Function Calls
    http://msdn.microsoft.com/en-us/library/bb288454.aspx

CWE-676: Use of Potentially Dangerous Function

- Function that can be used safely, but is often used in a way leading to a vulnerability
  - E.g. using `strcpy(destination, source)` without verifying `destination` buffer is large enough
  - https://docs.microsoft.com/en-us/previous-versions/bb288454(v=msdn.10)
  - https://github.com/intel/safestringlib/wiki/SDL-List-of-Banned-Functions
- Identifiable by text search, prepared header files ("pragma deprecated"), static analysis

CWE-676: Use of Potentially Dangerous Function

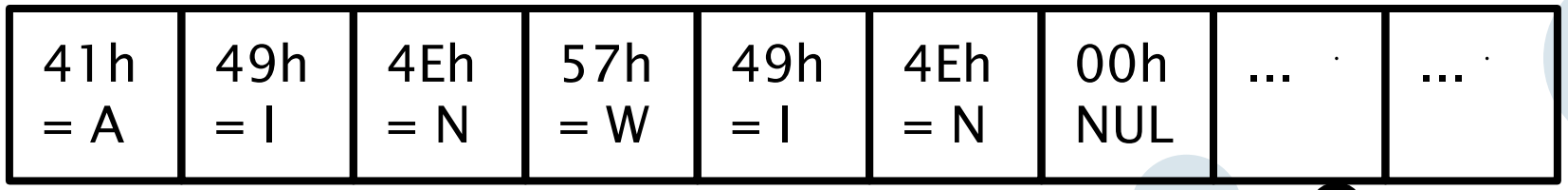
- strcpy, strcpyA, strcpyW, wcscpy, _tcscpy, _mbscpy, StrCpy, StrCpyA, StrCpyW, lstrcpy, lstrcpyA, lstrcpyW, _tccpy, _mbccpy, _ftcscpy
- strcat, strcatA, strcatW, wcscat, _tcscat, _mbscat, StrCat, StrCatA, StrCatW, lstrcat, lstrcatA, lstrcatW, StrCatBuff, StrCatBuffA, StrCatBuffW, StrCatChainW, _tccat, _mbccat, _ftcscat
- sprintfW, sprintfA, wsprintf, wsprintfW, wsprintfA, sprintf, swprintf, _stprintf)
- wvsprintf, wvsprintfA, wvsprintfW, vsprintf, _vstprintf, vswprintf
- strncpy, wcsncpy, _tcsncpy, _mbsncpy, _mbsnbcpy, StrCpyN, StrCpyNA, StrCpyNW, StrNCpy, strcpynA, StrNCpyA, StrNCpyW, lstrcpyn, lstrcpynA, lstrcpynW
- strncat, wcsncat, _tcsncat, _mbsncat, _mbsnbcat, StrCatN, StrCatNA, StrCatNW, StrNCat, StrNCatA, StrNCatW, lstrncat, lstrcatnA, lstrcatnW, lstrcatn
- gets, _getts, _gettws
- IsBadWritePtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadHugeReadPtr, IsBadCodePtr, IsBadStringPtr
- memcpy, RtlCopyMemory, CopyMemory, wmemcpy, lstrlen

C string handling

- Example scanf() ➔ sscanf_s

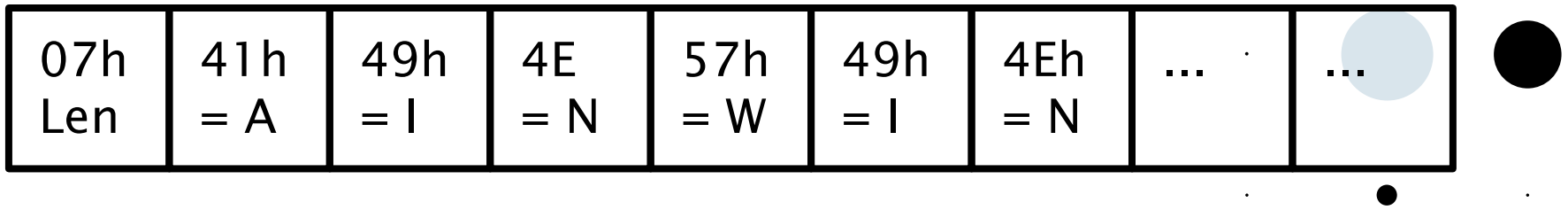
```
char buffer[7];
buffer[sizeof(buffer)-1] = '\0';
sscanf_s(buffer, "%s", sizeof(buffer));
```

- "AINWINAë!"
- String correctly terminated



String handling

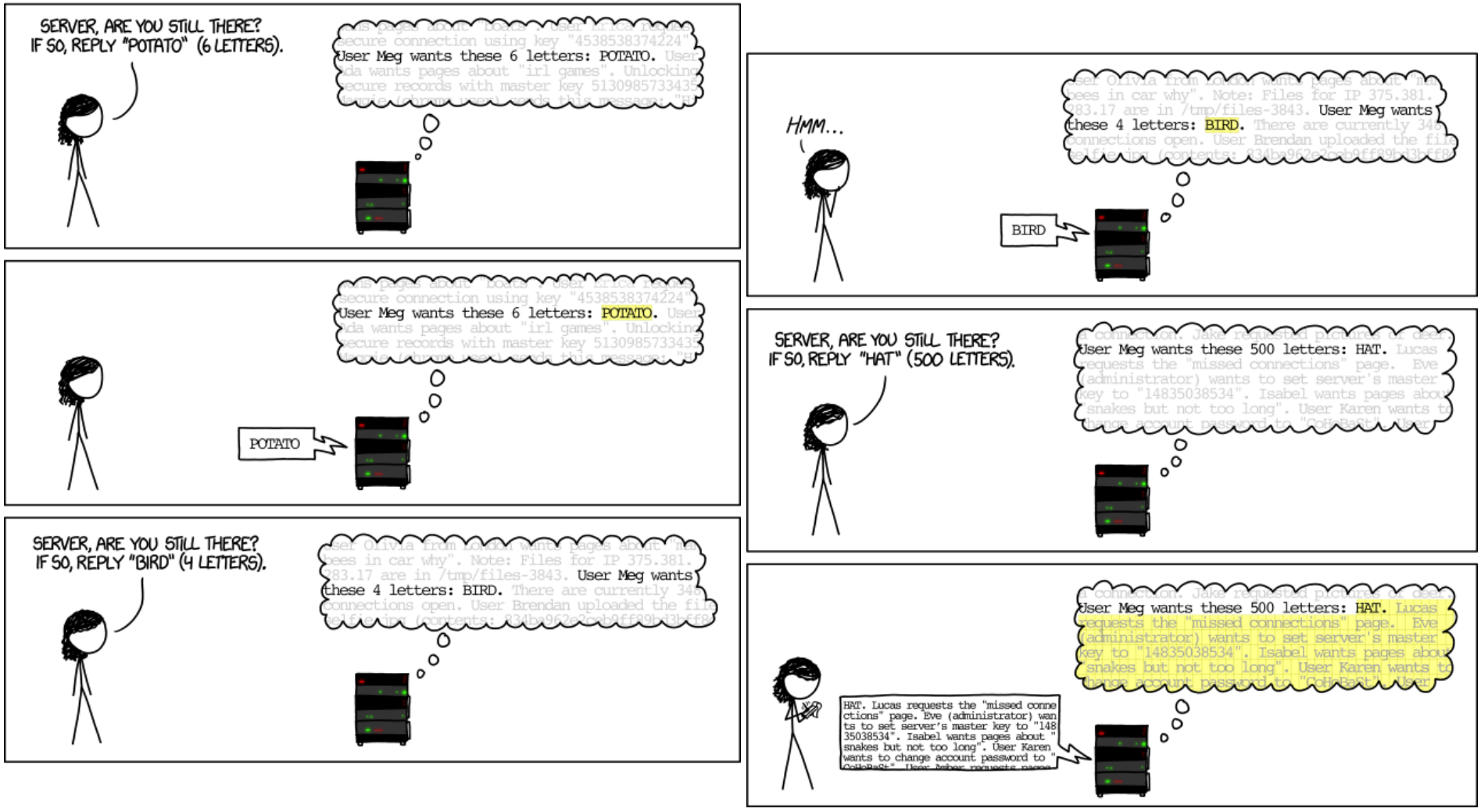
- Different internal string representation: pointer+length
  - Also allows to store NUL char
  - E.g., found in Pascal, Java
- ACMqueue article about effects of design decision to use ptr+nul byte instead of ptr+len to represent strings. Kamp (2011). The Most Expensive One-byte Mistake. http://queue.acm.org/detail.cfm?id=2010365



Defensive Programming

- Never assume anything
  - Only accept input from trustworthy callers as correct
  - Trusted ≠ Trustworthy
- Always validate input received from untrustworthy callers
  - Size, format, sequence, integrity, authenticity
- Always verify if needed resources are available
- Anticipate failures and handle them

"Heartbleed" (CVE-2014-0160)



https://www.us-cert.gov/ncas/alerts/TA14-098A

"Heartbleed" (CVE-2014-0160)

```
void *memcpy(
   void *dest,
   const void *src,
   size_t count
);
```

```
+               /* Allocate memory for the response, size is 1 byte
+                * message type, plus 2 bytes payload length, plus
+                * payload, plus padding
+                */
+               buffer = OPENSSL_malloc(1 + 2 + payload + padding);
+               bp = buffer;
+
+               /* Enter response type, length and copy payload */
+               *bp++ = TLS1_HB_RESPONSE;
+               s2n(payload, bp);
+               memcpy(bp, pl, payload);
```

- `ssl/d1_both.c`, 2012-01-01
  https://git.openssl.org/gitweb/?p=openssl.git&a=commit&h=4817504d069b4c5082161b02a22116ad75f822b1

"Heartbleed" (CVE-2014-0160)

```
void *memcpy(
   void *dest,
   const void *src,
   size_t count
);
```

```
+            /* Allocate memory for the response, size is 1 byte
+             * message type, plus 2 bytes payload length, plus
+             * payload, plus padding
+             */
+            buffer = OPENSSL_malloc(1 + 2 + payload + padding);
+            bp = buffer;
+
+            /* Enter response type, length and copy payload */
+            *bp++ = TLS1_HB_RESPONSE;
+            s2n(payload, bp);
+            memcpy(bp, pl, payload);
```

- `ssl/d1_both.c`, 2012-01-01
  https://git.openssl.org/gitweb/?p=openssl.git&a=commit&h=4817504d069b4c5082161b02a22116ad75f822b1

"Heartbleed" (CVE-2014-0160)

```
n2s(p, payload);
```

Read 2 bytes from p and store 16 bit value in payload

```
1462        -   /* Read type and payload length first */
1463        -   hbtype = *p++;
1464        -   n2s(p, payload);
1465        -   pl = p;
1466        -
1467  1462      if (s->msg_callback)
1468  1463          s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
1469  1464              &s->s3->rrec.data[0], s->s3->rrec.length,
1470  1465              s, s->msg_callback_arg);
1471  1466
      1467  +   /* Read type and payload length first */
      1468  +   if (1 + 2 + 16 > s->s3->rrec.length)
      1469  +       return 0; /* silently discard */
      1470  +   hbtype = *p++;
      1471  +   n2s(p, payload);
      1472  +   if (1 + 2 + payload + 16 > s->s3->rrec.length)
      1473  +       return 0; /* silently discard per RFC 6520 sec. 4 */
```

- ssl/d1_both.c, 2014-04-06
  https://github.com/openssl/openssl/commit/96db9023b881d7cd9f379b0c154650d6c108e9a3

```
void *memcpy(
   void *dest,
   const void *src,
   size_t count
);
```

```
+   /* Allocate memory for the response, size is 1 byte
+    * message type, plus 2 bytes payload length, plus
+    * payload, plus padding
+    */
+   buffer = OPENSSL_malloc(1 + 2 + payload + padding);
+   bp = buffer;
+
+   /* Enter response type, length and copy payload */
+   *bp++ = TLS1_HB_RESPONSE;
+   s2n(payload, bp);
+   memcpy(bp, pl, payload);
```

- `ssl/d1_both.c`, 2012-01-01
  https://git.openssl.org/gitweb/?p=openssl.git&a=commit&h=4817504d069b4c5082161b02a22116ad75f822b1

User Input

Input processing

- Behaviour of application determined by
  - Code
  - Internal state (memory content)
  - **Input**
- Insufficient checking and validation of input may lead to unexpected control flow

Input sources (UI)

- Keyboard, mouse, touch, button, voice, gestures, camera, pen
- Local human user, remote human user
- Clipboard data
- Simulated by processes (e.g. malware)

- Assistive technology for handicapped users

https://en.wikipedia.org/wiki/QWERTY#/media/File:QWERTY_keyboard.jpg

Trustworthiness of user input

- Who can **create** user input?
- Who can **modify** user input?
- Who can **select** user input?
- How can we **authenticate** user input?
- (Who can **observe** user input?)

Challenges

- **Banking:**
  Is this my bank's website?

- **PIN/Password:**
  Is my password input safe against keyloggers?

- **Cloud service:**
  Is this really the user I am talking to?

Output to user

- Help user to correctly operate
  - Prevent/detect fake login
  - Prevent/detect phishing
- Web Security Context: User Interface Guidelines
  http://www.w3.org/TR/wsc-ui/

Detection of automation/replay

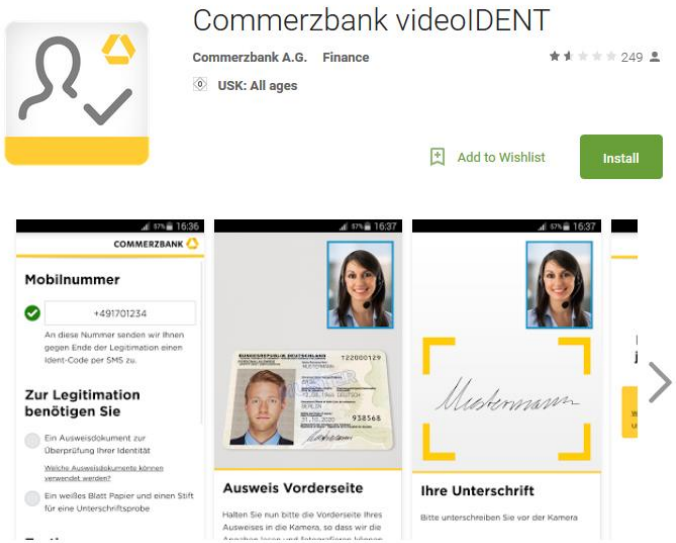
- PIN pad permutation
  - Show numbers on PIN pad in different order
  - Order visible to user, not discernible for malware
  - E.g., enter "4293" for PIN "1234"
- Simulated input yields wrong PIN

Preventing automation

- CAPTCHA
  - **Completely Automated Public Turing** Test to tell **Computers** and **Humans Apart**
  - Background: Turing, A.M. Computing machinery and intelligence. Mind 59, 236 (1950), 433–460 https://www.cs.mcgill.ca/~dprecup/courses/AI/Materials/turing1950.pdf
  - http://www.captcha.net/
- Visual challenge easily solved by humans, hard for algorithms (at present)
- **Usability questionable**

Video and voice

- Hard to evaluate
- Hard to create, easier to replay
- Examples:
  - "Netswipe"
    (show credit card to webcam)
  - "VoiceProof" (PIN input by voice)
  - "videoIDENT"
    (show ID card)



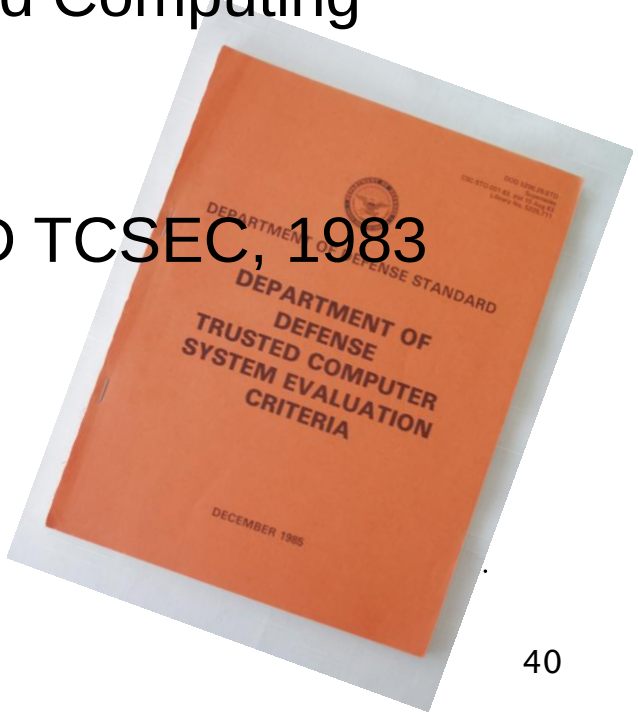
https://www.engadget.com/2011/07/26/netswipe-turns-your-webcam-into-a-credit-card-reader-brings-pos/

Trusted Path

Trusted Path

– «Trusted Path - A mechanism by which a person at a terminal can communicate directly with the Trusted Computing Base. This mechanism can only be activated by the person or the Trusted Computing Base and cannot be
imitated by untrusted software.»

— DoD TCSEC, 1983



https://en.wikipedia.org/wiki/Trusted_Computer_System_Evaluation_Criteria#/media/File:Orange-book-small.PNG

Trusted path

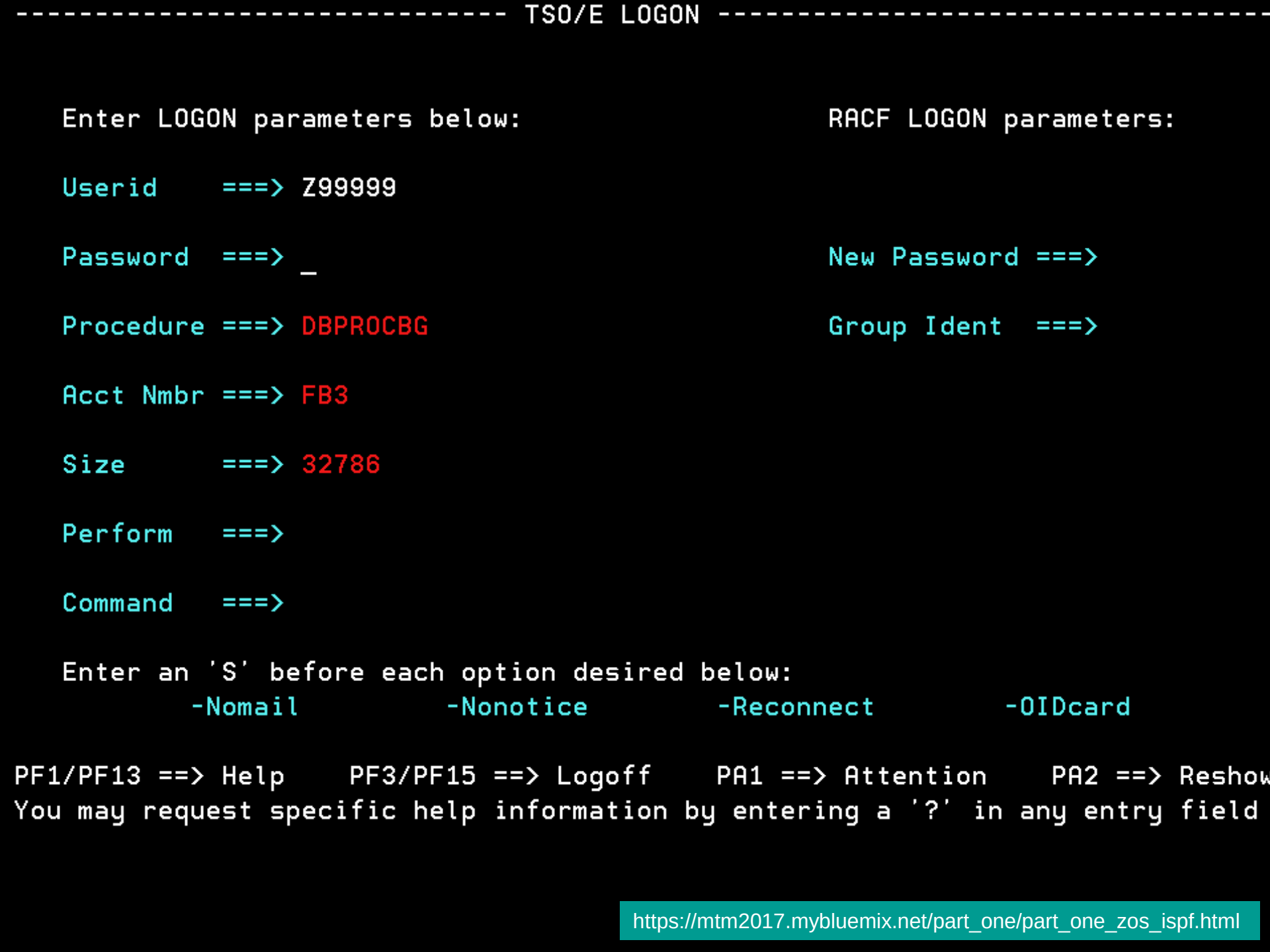
- Examples:
  - **Ctrl+Alt+Del** to reach logon screen on isolated desktop
  - Terminal console **physically** at server
  - **Mobile phone display** (under assumption that no malware executed on phone)
- Trusted path for an application
  - "Person at a terminal" = **user**
  - "Trusted Computing Base" = **application**

Hardware trusted path

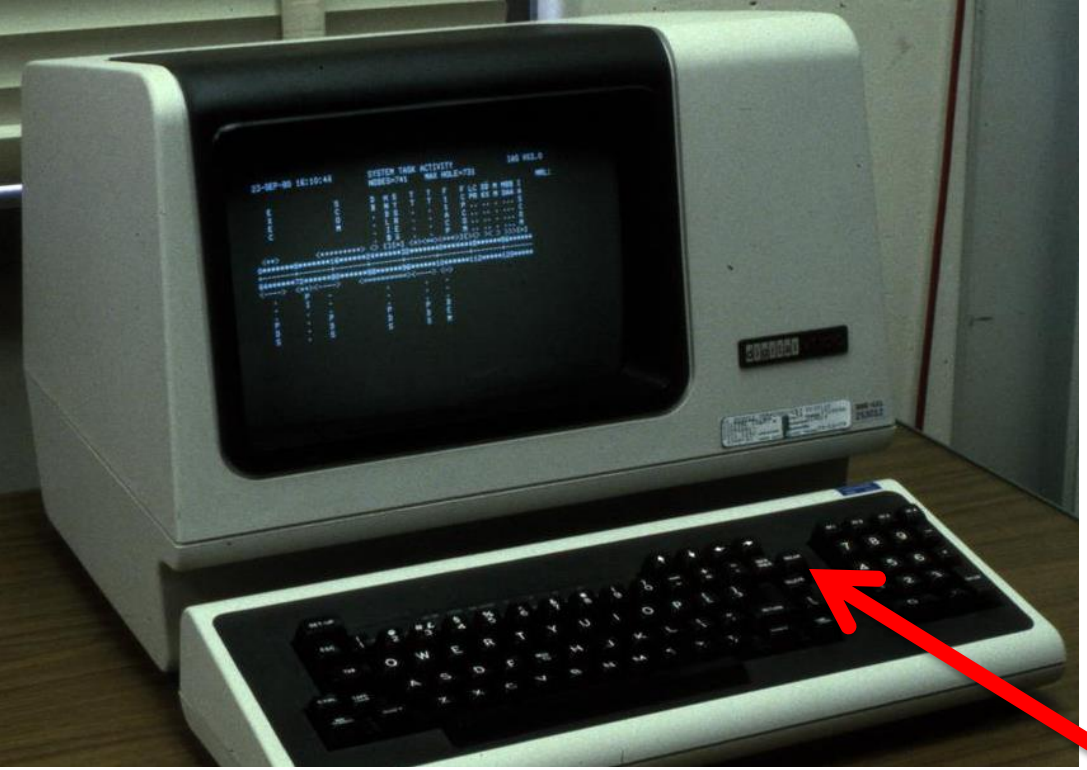
- Dedicated device, single purpose
  - Display accessed only by internal code
  - Keypad accessed only by human user
  - Authentication: possession, PIN, password, biometrics
  - Expensive
- Does not scale for
  # applications

42

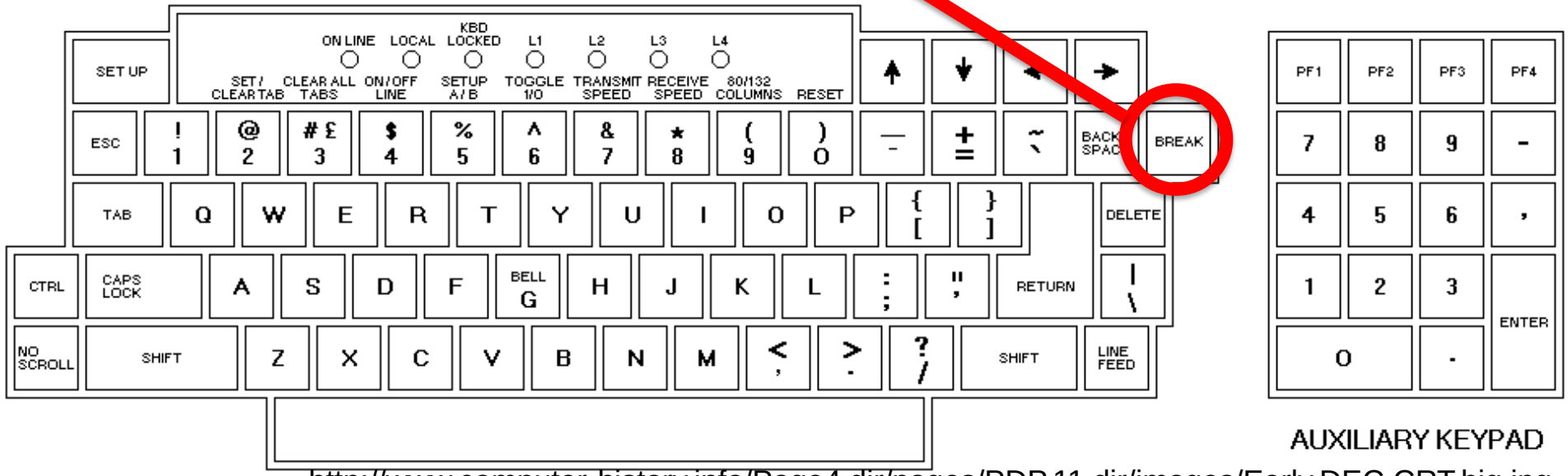
https://www.reiner-sct.com/ccsdata/attImage.png?attachmentId=79543



https://mtm2017.mybluemix.net/part_one/part_one_zos_ispf.html



BREAK key
terminates connection to mainframe, forces **logon on re-connect**



http://www.computer-history.info/Page4.dir/pages/PDP.11.dir/images/Early.DEC.CRT.big.jpg

MULTICS logout

I&A. Second, it disallows making use of the "logout -hold" feature to return the user to the I&A dialog. This installation parameter is the mechanism used to enforce the use of a "trusted path". On Multics, the only assurance one has that the I&A dialog is undertaken with trusted software (the answering service) as opposed to with a trojan horse, is after first having broken the communications channel connection and reconnected it. A site may wish to ensure

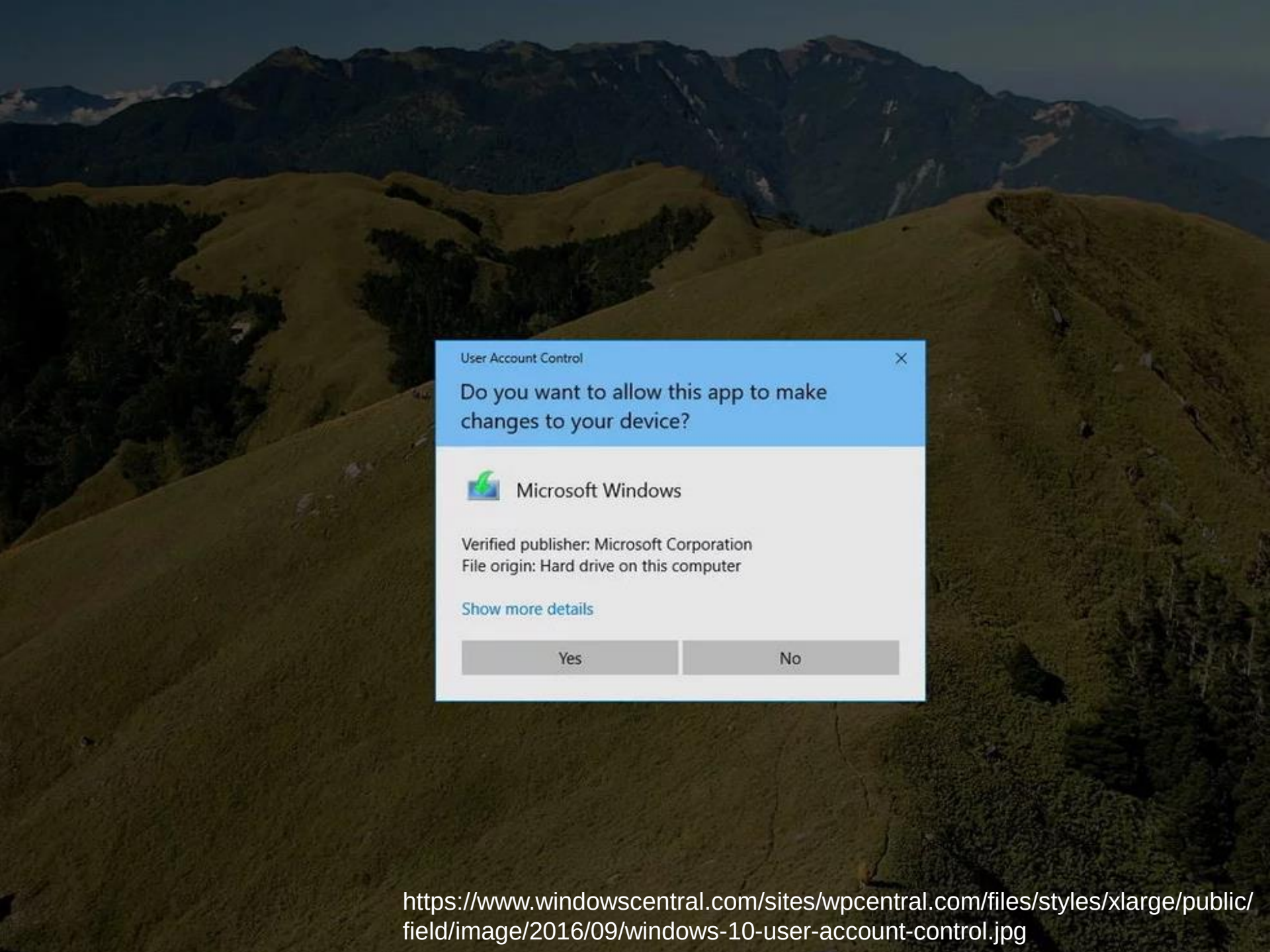
http://web.mit.edu/multics-history/source/Multics/mdds/mdd010.compout

http://web.mit.edu/multics-history/source/ldd_listings/sss/logout.list

```
149       if logout_string.hold
150       then do;
151            string (trusted_path_flags) = system_info_$trusted_path_flags ();
152            if trusted_path_flags.login then do;
153               if logout_string.hold & my_name = "logout" then do;
154                    call com_err_ (0, my_name, "logout -hold is not permitted at
155                    return;
156               end;
157            end;
158       end;
```

Software trusted path

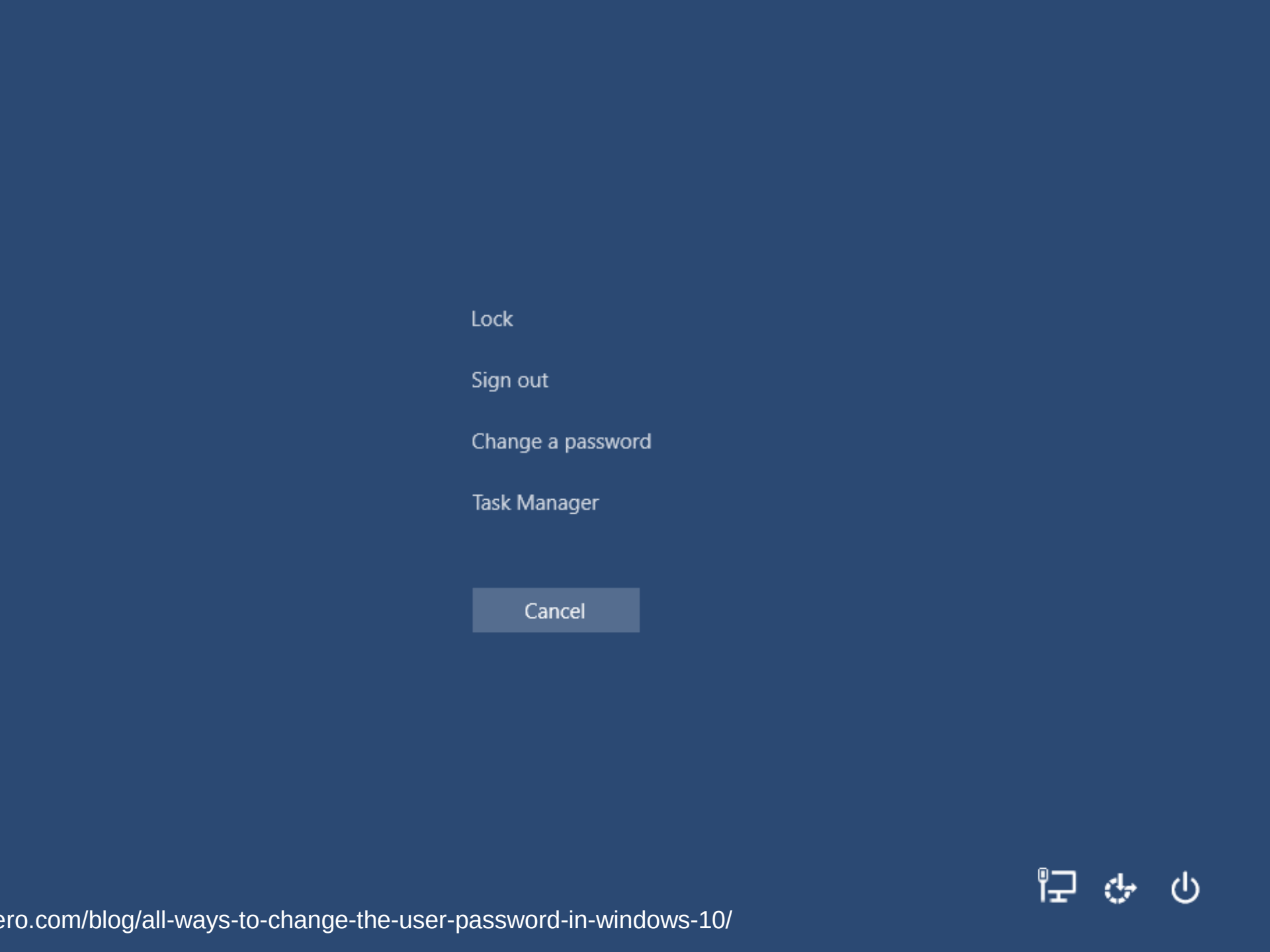
– UAC consent/credential prompt



https://www.windowscentral.com/sites/wpcentral.com/files/styles/xlarge/public/field/image/2016/09/windows-10-user-account-control.jpg

Software trusted path

- UAC consent/credential prompt
  - Run programs with administrative privileges
  - "Run as" **service**
  - Switch to **separate desktop**, not accessible by user processes (enforced by **ACL**)
  - Ask for confirmation (or request credentials)
- Switch back to user desktop
- Understanding and Configuring User Account Control in Windows Vista: http://technet.microsoft.com/en-us/library/cc709628(WS.10).aspx

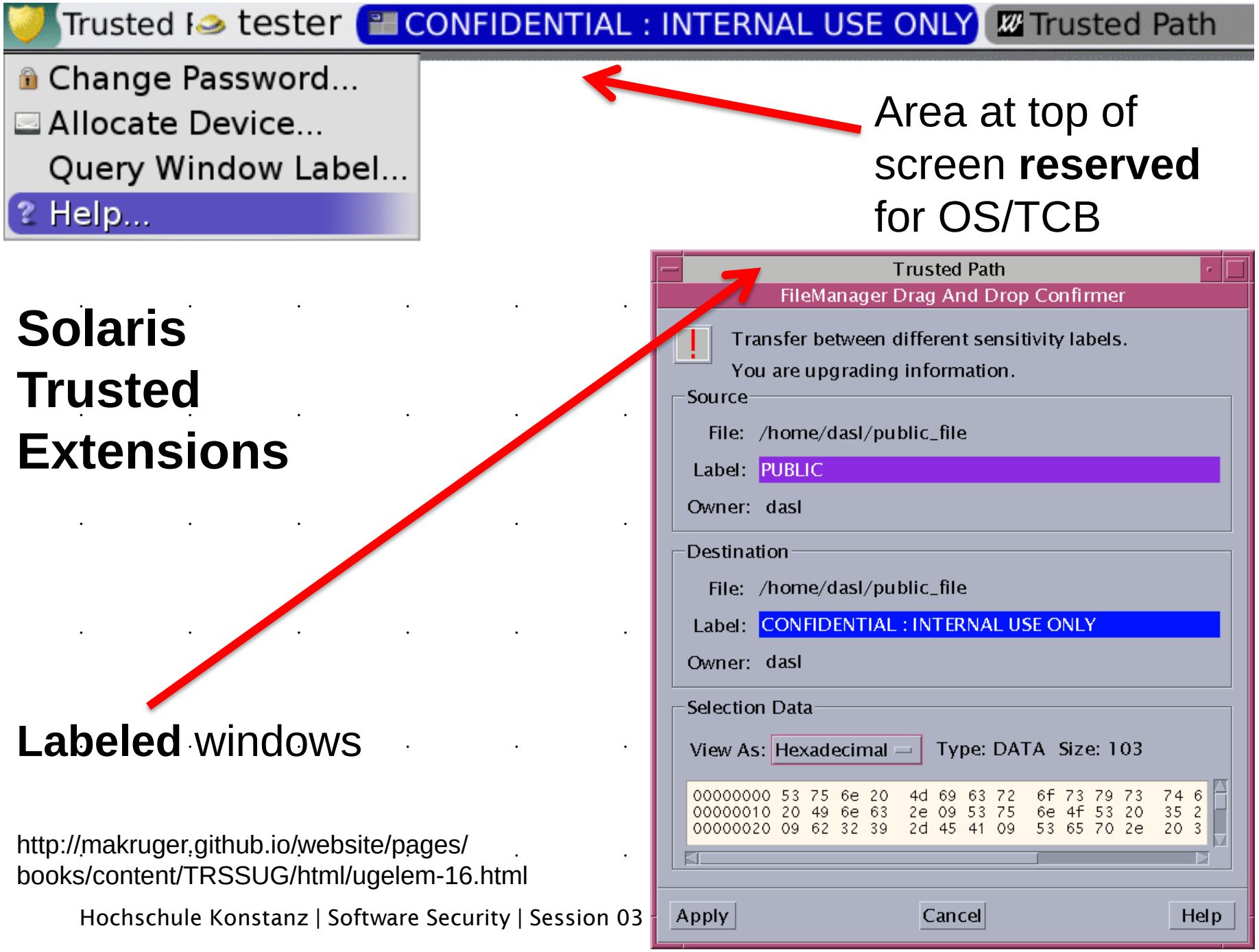


ero.com/blog/all-ways-to-change-the-user-password-in-windows-10/

WinSta0\Winlogon desktop ACL

Subject	Access mask
BUILTIN\Administrators	_DELETE
(SID: S-1-5-32-544)	DESKTOP_ENUMERATE
	READ_CONTROL
	WRITE_DAC
	WRITE_OWNER
NT AUTHORITY\SYSTEM	_DELETE
(SID: S-1-5-18)	DESKTOP_READOBJECTS
	DESKTOP_CREATEWINDOW
	DESKTOP_CREATEMENU
	DESKTOP_HOOKCONTROL
	DESKTOP_JOURNALRECORD
	DESKTOP_JOURNALPLAYBACK
	DESKTOP_ENUMERATE
	DESKTOP_WRITEOBJECTS
	DESKTOP_SWITCHDESKTOP
	READ_CONTROL
	WRITE_DAC
	WRITE_OWNER

- Desktop object as security boundary
- Restricted which processes run on the Winlogon desktop



Solaris Trusted Extensions

Labeled windows

http://makruger.github.io/website/pages/books/content/TRSSUG/html/ugelem-16.html

Mobile phones

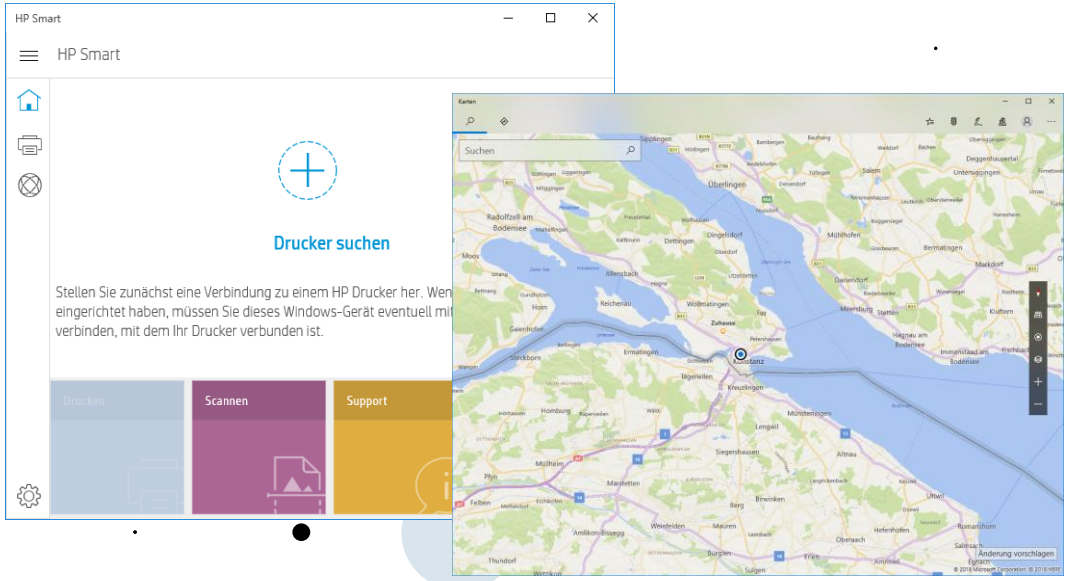
- **Single app UI**, moving away from windowed UI
  - Sandboxed apps
  - No interference from other apps
  - Split screen controlled by user
- **Home button** to invoke trusted path
  - Only OS reacts
  - App is suspended/ terminated
  - Home screen content defined by OS

https://cnet4.cbsistatic.com/img/PjH62hzziT5gVuNIxs902hhPl6Q=/1600x900/2017/10/13/0214aab0-2ff7-4de0-be74-e8f565a29d60/windows-10-phone-still.jpg

Windows 8/10 Modern UI

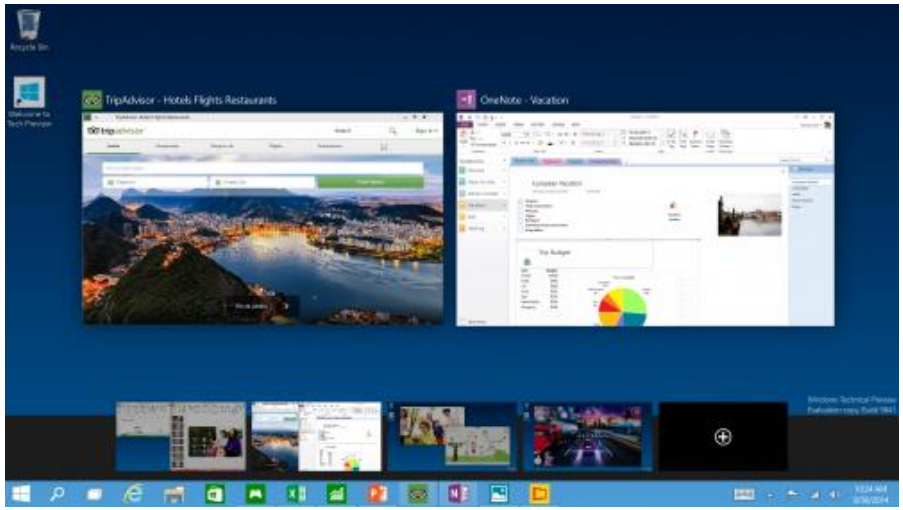
- **Sandboxed** execution of apps
- **Reserved screen area** for app, full screen/side-by-side
- Similar to mobile phone UI

- Integrity (?)
- Authenticity (-)
- Confidentiality (-)



Windows 10 multiple desktops

- No real desktops with ACLs
- Useful to organize windows, **not to limit access** to applications



https://en.softonic.com/articles/windows-10-multiple-desktops

- Integrity (?)
- Authenticity (-)
- Confidentiality (-)

Window station

- **Securable object**, i.e., has ACL
  - Clipboard, atom table, 1..n desktop objects
- `Winsta0`: **interactive** window station
  - Only WS that can display UI or receive user input
- Assigned to logon session of interactive user
- All other window stations non-interactive
- Terminal services: each session with own interactive window station
  (Integrated as "Fast User Switching" since XP)

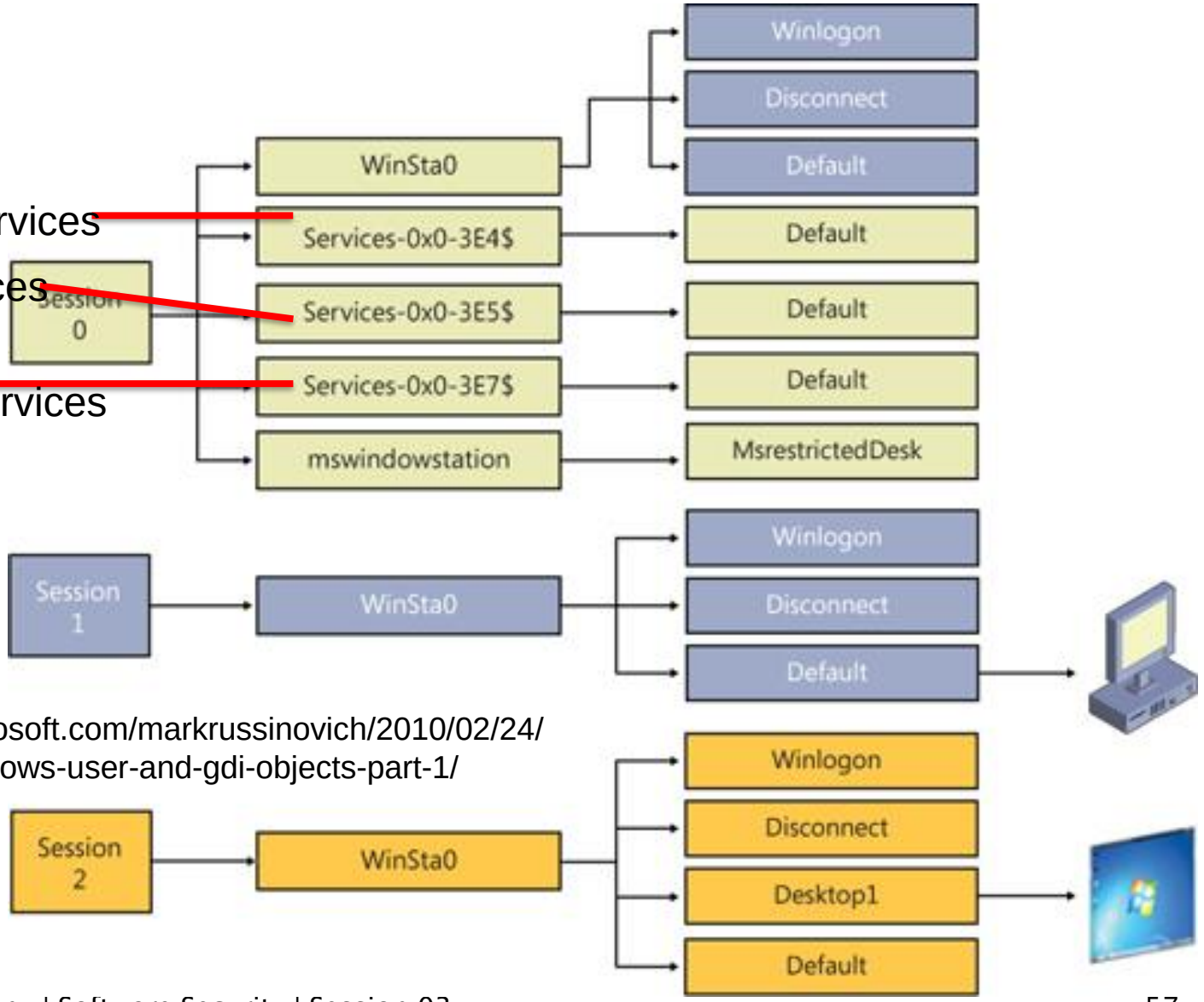
Desktop

- **Securable object**, i.e., has ACL
  - Logical display surface, can span several monitors
  - Contains UI objects: windows, menus, hooks etc.
  - **Shared** display surface
- Window **messages only** between threads **on same desktop**
- Only one desktop active at a time
  - `Winsta0`: Default, Screen-saver, Winlogon
- Richards/Beeder/Larsen (2013). Sysinternals Desktops. http://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-32-Desktops

NetworkService services

LocalService services

LocalSystem non-interactive services



https://blogs.technet.microsoft.com/markrussinovich/2010/02/24/pushing-the-limits-of-windows-user-and-gdi-objects-part-1/

UIPI User Interface Privilege Isolation

- 3 levels for processes: low, medium, high
  - Interaction at **same or lower** levels
  - **No Window messages** sent to higher levels
  - **No hooks** to monitor higher levels
  - **No DLL injection** into higher levels
- Windows Integrity Mechanism Design
  http://msdn.microsoft.com/en-us/library/bb625963.aspx

Job objects

- **Securable object**, i.e., has ACL
  - Manage a **group of processes**
  - Operations on job object **affect all** processes
- JOBOBJECT_BASIC_UI_RESTRICTIONS
  - Limit creation/switching of desktops
  - Clipboard, atom table, handles
  - Display settings, UI configuration

Summary

- Architectural risk analysis to discover
  - **Typical weaknesses**
  - **Ambiguity** that could lead to unwanted program behviour
  - **Dependencies** that require trustworthiness of environment
- **Always validate input** from untrustworthy sources
  - Input sources might not be obvious
  - Trustworthiness might not be obvious
- Ensure user input cannot be captured/fabricated/replayed
  - Requirements vary based on purpose of user input
  - Not all user input is security-sensitive