Marko Laakso is a loony Finn with UNIX background, gained from administrating the network of the Department of Electrical Engineering at the University of Oulu. AusCERT advisories from summer 1996 pushed him beyond point of no return by raising the question "More of this same kind, again?" Ever since he has been pestering AusCERT and vendors with bug reports too long to be read.

Ari Takanen, a complementary reinforcement with NT networking and administration background, brings in his expertise and interests in the field. He is set out to prove that there are similarities in vulnerabilities beyond operating system boundaries. His goal is at the Diploma Engineer Degree (MSEE), meanwhile he works as a Research Scientist at the Department of Electrical Engineering.

Since 1983 Professor Juha Röning has been a member of faculty of the Department of Electrical Engineering at the University of Oulu, where he received his Diploma Engineer Degree (MSEE), 1983, Licentiate in Technology with Honours, 1985, and Doctor of Technology, 1992. He is a member of SPIE, IEEE, Sigma Xi, Finnish Pattern Recognition Society, and Finnish Artificial Intelligence Society (FAIS).

Together they form the Secure Programming Group at the University of Oulu, Finland (OUSPG). Their main objective is to promote proactive vulnerability work.

The vulnerability process: a tiger team approach to resolving vulnerability cases

Marko Laakso, Ari Takanen, Juha Röning
University of Oulu, Department of Electrical Engineering
Computer Engineering Laboratory
Linnanmaa BOX 4500, FIN-90401 Oulu, Finland
{fenris,art,jjr}@ee.oulu.fi

ABSTRACT

Security vulnerabilities that affect widely deployed software emerge frequently. Addressing these maladies requires coordination and communication. The purpose of this work was to explore a systematic approach to reporting and resolving vulnerability cases, from a tiger team perspective. A life-cycle model with checkpoint-based metrics was developed, a case study was carried out and observations were gathered. The vulnerability process was found to be complicated but manageable. The role of the FIRST teams as coordinators is encouraged and a shift from undisciplined, reactive vulnerability work towards a professional, proactive approach is promoted.

1. Introduction

A software system, whether an application or an operating system, may contain security-related flaws that lead to loss of confidentiality, integrity or availability of the information resources. These flaws introduce vulnerabilities that may be exploited in ways that violate the security policy of the system. Software vulnerabilities that affect widely deployed software packages are brought forward in public forums on a weekly basis. In addition to problems with brand new products, fresh vulnerabilities emerge even from packages that have otherwise withstood the test of time. The overall impact of deficiencies in the software itself goes beyond the effect of vulnerabilities caused by improper configurations of individual installations.

Software vulnerabilities are disclosed in varying fashions. These include public disclosures, security advisories and security bulletins from vendors. Vulnerability reports may appear on full disclosure mailing lists and various distribution lists, and sometimes even in mainstream media. New vulnerabilities are found by the vendors themselves, and also by their customers and independent organisations when evaluating the security of the subject software. Vulnerabilities manifest themselves during security reviews, quality assurance and normal system operation, and sometimes in more thorough penetration testing. As part of its research, the Secure Programming Group at the University of Oulu (OUSPG) has adopted a role of the vendor independent security evaluator, a software vulnerability tiger team [1]. OUSPG has been a source of several vulnerability reports.

The life-cycle of a vulnerability case begins with the discovery of the defect and terminates after all aspects of the problem have been properly resolved. Communication is required between different actors with potentially conflicting roles. These actors, roles and activities form the vulnerability process, which ultimately aims to complete the life-cycles of the vulnerability cases in an orderly manner.

Different points of view regarding the vulnerability process have been presented amongst security professionals. The vendor perspective has been highlighted at the vendor panel coordinated by Sanchez at a Forum of Incident Response and Security Teams (FIRST) conference in 1997 [2]. In 1996, at a FIRST workshop, McMillan described vulnerability handling procedures and advisory team activities from the viewpoint of computer emergency response teams [3].

There is a lack of documentation and support aimed at vendor-independent parties discovering novel software vulnerabilities, and this has resulted in less than optimal life-cycles for vulnerability cases and an undisciplined process that has been hard to understand, control and improve.

The purpose of the work reported here was to support teams and private persons who unveil security vulnerabilities while evaluating, reviewing and testing software products. The material provided herein aims to encourage a disciplined and analytic approach to reporting and resolving vulnerability cases, and to highlight practical experiences gained from the perspective of a vulnerability tiger team. An attempt is made to improve the vulnerability process both by providing both metrics for gathering performance statistics and by describing observations of potential problem areas.

We introduce here a model for the vulnerability life-cycle, and provide supporting terminology. Different actors and their roles are identified, primitive checkpoint-based metrics are described, a simple case study is carried out and observations from different phases of the vulnerability process are presented, both based on the OUSPG case history. Possible avenues for continuing and expanding vulnerability work are discussed and conclusions are drawn.

2. The vulnerability process

Analysis and documentation of the context and internals of vulnerability work are prerequisites for understanding, controlling and improving the process of resolving vulnerability cases. We introduce here first the terminology and later the approach used by the OUSPG, including the models and metrics adopted.

2.1. Terminology

Uniform terminology is a necessity for efficient communication and documentation. A study of the terminology used in the context of vulnerability analysis was presented by Krsul [4]. Definitions of the roles, actors and activities used in this study will follow in the subsequent sections. OUSPG adaptations of central terms and their definitions are set out below:

Errors made during software development result in *faults* that may lead to *failures*, i.e. incorrect or undesired program behaviour [5].

A possibility of a failure in a security-critical portion of an application is manifested as *a potential vulnerability*. If an actual method for triggering an insecure state in the application is known, then the application is considered to be *exploitable* via a verified vulnerability. *A verified vulnerability*, designated later as *a vulnerability*, exposes information resources to the loss of confidentiality, integrity or availability. *Reproduction* of a vulnerability occurs when the vulnerability is verified by an another party after receiving the vulnerability details.

The term *exploit* is used to denote either a passive or an active attack method or receipt aimed to take advantage of an existing vulnerability. Using an exploit to perform an attack to violate the security policy of the target system will be referred to as *exploitation* of the vulnerability.

The term *vulnerability case* is used to refer to the basic processing unit in vulnerability work. A vulnerability case may contain only one specific vulnerability, or it may group together several related vulnerabilities. The term *vulnerability life-cycle* is used to designate the emergence of a vulnerability case, its evolutionary phases and termination. Vulnerability work, referred to later on as the *vulnerability process*, carries out the life-cycles and engages in activities such as vulnerability prevention, feedback integration and efficiency improvement. This study excludes patch deployment from the vulnerability process.

A vulnerability case is initiated by a *vulnerability disclosure*. In an *internal disclosure* the vulnerability details are in the possession of parties directly involved in the vulnerability process, as determined by the *need-to-know* principle. A *public disclosure* is said to have taken place when the details are known by a wider audience. Active sources for public disclosures are *full disclosure mailing lists* such as BugTraq and NTBugTraq [6] [7]. These lists are email distributions that promote the dissemination of vulnerability details to all interested parties. If a public disclosure is sought for, then the *grace period* is the amount of time given to the affected vendor to react before the details are published.

Information about a vulnerability case may be disseminated in a *vulnerability advisory*. This is supported by an *impact assessment*, which determines the seriousness of the vulnerability by considering who is affected and how. An interim *work-around* or a software fix, known as a *patch*, addresses the vulnerability by either limiting or preventing its exploitation.

Software penetration testing is an activity aimed at disclosing vulnerabilities in target software, i.e. in the selected *subjects*. A vulnerability case follows the *penetrate-and-patch paradigm* when the vulnerability is found by penetration testing and addressed by an administrative work-around or software correction. *A vulnerability tiger team* is a person, group or organisation that conducts software penetration testing in order to assess the security of the subjects. The penetration testing may be carried out by monitoring during normal operation, casual inspection, formal evaluation or systematic testing.

2.2. Actors and their roles

Three major roles have been identified in the vulnerability process (Figure 1) and are described here in detail. Moreover, two supplementary roles are noted. These may be adopted by actors such as the vulnerability tiger teams, vendors and security organisations.

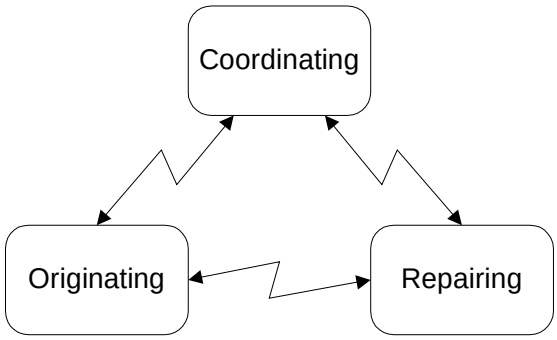

Figure 1: Major roles inside the vulnerability process

An actor in the *originating role* is the source of the vulnerability report that initiates or supplements a case life-cycle. An *originator* discovers the vulnerability during penetration testing. Actors in this role have expertise in the security problem domain and thus should participate in developing an interim work-around, evaluating the final fix and reviewing a potential advisory.

An actor in the *coordinating role* supervises the vulnerability work by assisting the other participants in carrying out the vulnerability case life-cycles and by accumulating expertise in the process domain. A *coordinator* assists in establishing and maintaining the communication link between the originators and repairers. The communication will benefit from existing security contacts maintained by the coordinators. Actors in this role may provide vulnerability verification and impact assessment and may direct the life-cycle activities accordingly. Coordinators may filter and combine the vulnerability information if several originators are involved in the case. Vulnerability characteristics may require a call for the involvement of several repairers. The coordinators may evaluate and develop administrative work-

arounds and participate in evaluating the final correction. They may provide the channels for distributing vulnerability information as widely as possible after production or evaluation of the advisory in co-operation with other parties.

An actor in the *repairing role* should be the expert in the application domain of the product affected by the vulnerability case. Ideally a *repairer* should be involved in the organisation responsible for developing and distributing the vulnerable product. The involvement of actors in this role is essential for developing and distributing the work-arounds and patches, and for assessing the impact of the vulnerability and the side-effects of the proposed solutions. Due to their familiarity with the use and user base of the vulnerable product, actors in this role should either initiate or evaluate the advisories addressing the issue.

In addition to the roles considered above, which are inherent in the vulnerability process, supplementary roles can be identified. The main deliverables of the process, improved security of the software products and heightened vulnerability awareness, are assumed to implicitly benefit the users of the products. Users represent an external entity in a *consuming role*. An actor in a *tracking role* may be a bystander collecting vulnerability information for research purposes or a party who contributes its expertise to a case without engaging in any of the major roles.

Individuals, groups and organisations participating in the vulnerability work may assume any of the roles introduced above. Two or more roles may be assumed by a single entity, e.g. a vendor with an engineering team in a repairing role and a vendor's security coordination team in a coordinating role (Example A).

Example A: A vulnerability in Product P is discovered by an external vulnerability tiger team (originator) during penetration testing. The vulnerability is reported directly to the respective Vendor V, where the information is forwarded to the Vendor's internal security coordination team (coordinator). Information is dispatched to an engineering team (repairer) responsible for the Product P.

Multiple roles may merge inside an entity, e.g. a vendor with product engineering team internally discovering the vulnerability and subsequently providing the fix. A straight-forward mapping between the roles and actors can be reached when an external vulnerability tiger team acts as the originator, a Computer Security Incident Response Team (CSIRT) assumes the coordinating role and the respective vendor takes the repairing role.

The number of actors involved in a vulnerability case directly contributes to the complexity of communication. A typical vulnerability case known as a *multi-vendor* vulnerability may involve several vendors with vulnerable products due to a coincidence or inherited code-base (Example B). Multi-coordinator and multi-originator cases are feasible and have been observed in practice.

Example B: A vulnerability in Product P for operating system O is disclosed on a full disclosure mailing list by an anonymous contributor (originator). FIRST Team T (coordinator) verifies the vulnerability and discovers that it affects several operating systems related to O due to common inheritance. Vulnerability information is dispatched to all respective vendors (repairers).

2.3. A vulnerability tiger team approach

The discussion that follows will reflect the vulnerability process from the perspective of a vulnerability tiger team in the originating role. The activities and mutual communications of the coordinators and repairers are treated in less detail and are abstracted as black boxes (Figure 2).

The internal operation of the tiger team can be roughly divided into two subactivities. Its *vulnerability research* is the source of discoveries that initiates new vulnerability cases, and the technical aspects of the vulnerability case are handled in this research. *Vulnerability management* consists of tracking, communication and feedback integration, the main objective being to ensure that the cases proceed through their life-cycles and are eventually terminated. Performance statistics should be gathered during vulnerability tracking.

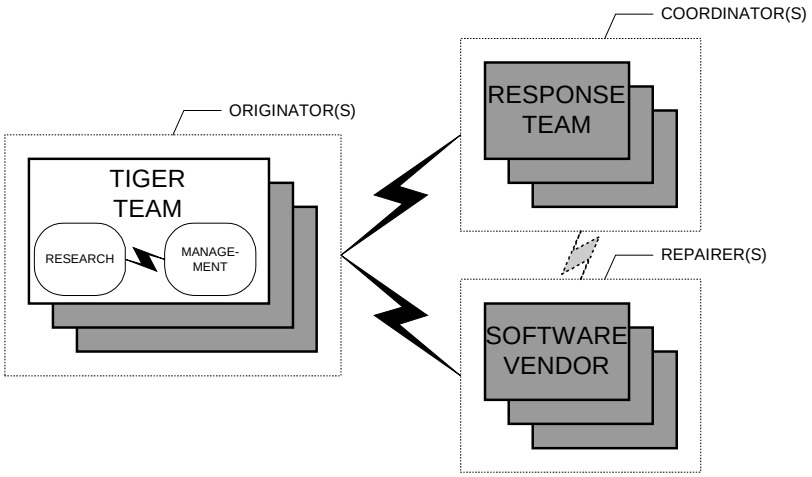

Figure 2: A tiger team in the originating role

2.4. A model of the vulnerability case life-cycle

The model of the vulnerability case life-cycle proposed here identifies the different stages that a case enters during its lifetime and the transitions between these stages (Figure 3).

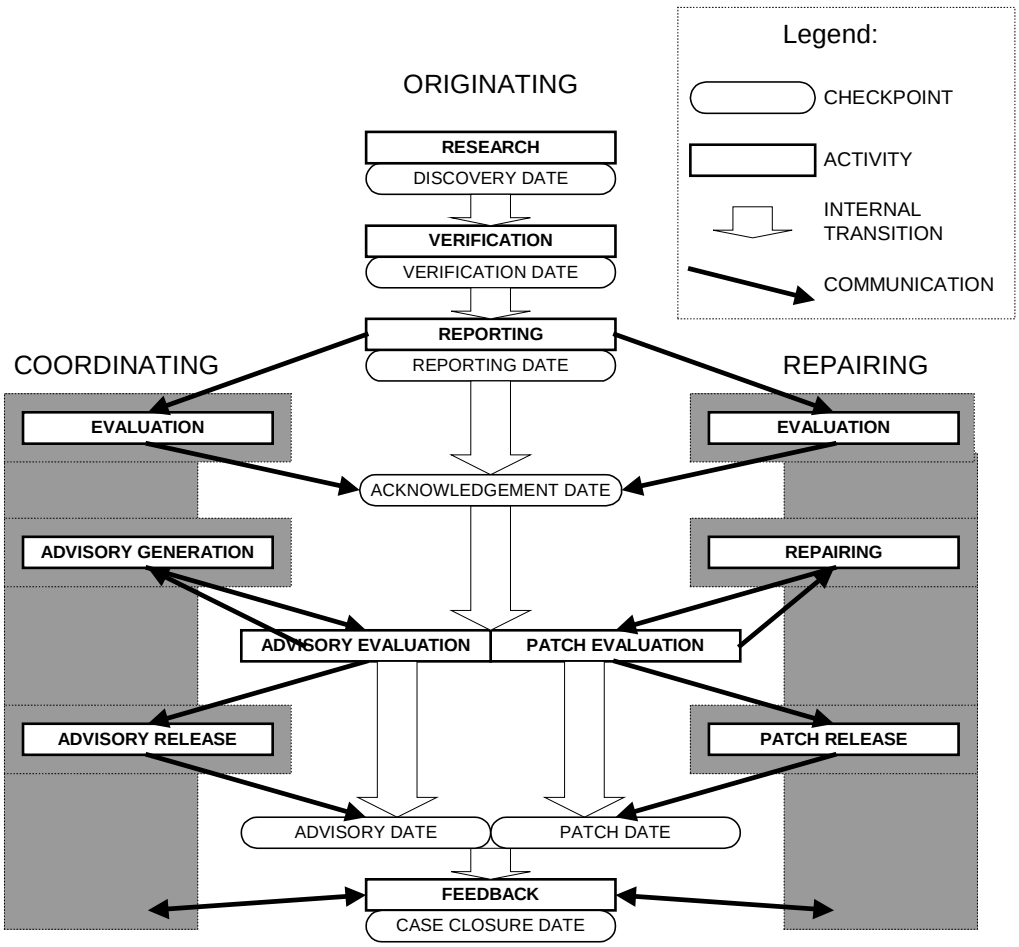

Figure 3: Model of a vulnerability life-cycle

At each stage a principal activity is carried out by actors in responsible roles. Each transition has a prerequisite in the form of a deliverable and represents a check-point that can be utilised for gathering performance statistics. Rejection of the vulnerability at certain stages will result in termination of the life-cycle.

Activity: Research
Responsible role: Originating
Deliverable: Potential vulnerability
Check-point: Discovery date

The research activity is the spontaneous entry point into the vulnerability life-cycle. Penetration testing or a comparable activity may result in a discovery that unveils a potential vulnerability.

Activity: Verification [optional]
Responsible role: Originating
Deliverable: Verified vulnerability
Check-point: Verification date

During verification the originating party attempts to filter false positives and assess the technical nature and exploitability of the potential vulnerability. If the originator lacks resources for verification, it may be sensible to proceed without a confirmation.

If the potential vulnerability is evaluated as definitely harmless, it may be rejected. If a successful verification takes place, the vulnerability details are supplemented by an exploit.

Activity: Reporting
Responsible role: Originating
Deliverable: Vulnerability report
Check-point: Reporting date

Essential information about the vulnerability is collected and recorded during the reporting phase. The report is delivered to the coordinators and to the identified repairers. Completeness of the vulnerability report is promoted in order to reduce the amount of communication.

An example of a form for a vulnerability report is presented in Appendix A. An outline for the vulnerability report should include:

- Tracking information
- Identification of the affected products and vendors
- Initial impact assessment
- Description of the test environment and subject software for reproduction purposes
- Technical description of the vulnerability
- Possible exploitation details (an exploit) for reproduction purposes
- Initial work-around if available
- Contact information

Activity: Evaluation [black box abstraction]
Responsible roles: Repairing and Coordinating
Deliverables: Acknowledgement, evaluation and change request
Check-point: Acknowledgement date

During the evaluation phase, the reported vulnerability is either acknowledged or rejected by the recipients. Rejection may be the result of a reproduction failure or an impact assessment conclusion. The vulnerability report combined with the evaluation results will initiate the repairing branch of the life-cycle via a change request. Impact assessment, work-around verification and observations from the reproduction form the basis for the advisory generation thread of the life-cycle. An acknowledgement or rejection notice is sent to the originators as feed-back. All parties should agree upon the case closure before final rejection.

Activity: Repairing [black box abstraction]
Responsible role: Repairing
Deliverable: Patch candidate
Check-point: None specified

A patch candidate is created as a fix proposal to address the vulnerability in the software.

Activity: Patch evaluation
Responsible roles: Coordinating, Originating and Repairing
Deliverable: Patch feedback
Check-point: None specified

A patch candidate is evaluated for effectiveness and correctness.

Activity: Patch release [black box abstraction]
Responsible role: Repairing
Deliverable: Released patch
Check-point: Patch date

The tested and accepted patch candidate is released.

Activity: Advisory generation [black box abstraction]
Responsible role: Coordinating
Deliverable: Draft advisory
Check-point: None specified

A draft advisory is written.

Activity: Advisory evaluation
Responsible roles: Coordinating, Originating and Repairing
Deliverable: Advisory feedback
Check-point: None specified

The draft advisory is evaluated for correctness.

Activity: Advisory release [black box abstraction]
Responsible role: Coordinating
Deliverable: Advisory release
Check-point: Advisory date

The accepted advisory is released.

Activity: Feedback
Responsible roles: Coordinating, Originating and Repairing
Deliverable: None specified
Check-point: Date of the case closure

During the feedback phase, observations on the case are exchanged and process statistics are updated. Feedback integration should support vulnerability prevention and the improvement of process efficiency.

2.5. Primitive metrics based on process check-points

Primitive metrics for measuring the performance of the vulnerability process were developed based on the life-cycle model and identified check-points. The unit for the proposed metrics is a time interval counted in days. The metrics and their abbreviations and definitions are illustrated in Table 1.

Table 1: Performance metrics

Metric	Abbrev.	Definition
Verification Delay	VD	Delay between discovery and verification
Reporting Delay	RD	Delay between verification and reporting
Evaluation Delay	ED	Delay between reporting and acknowledgement
Patch Delay	PD	Delay between reporting and patch release
Advisory Delay	AD	Delay between reporting and advisory release
Curing Delay	CD	MAX(PD, AD)
Treatment Delay	TD	VD + RD + CD

The internal performance of the originator can be measured by combining the verification delay (VD) and reporting delay (RD). The external performance of both the coordinators and repairers is characterised by the curing delay (CD). Overall performance of the process is driven by the treatment delay (TD).

3. A vulnerability case study

The OUSPG has been gathering tracking information for vulnerability cases since autumn 1996. Cases originating from the OUSPG have been recorded in full detail in accordance with the proposed model, and available data for appropriate reference cases have been entered in the database in order to support vulnerability research. A subset of 28 cases was used in the case study and was extracted from the database of 61 cases. The case study comprised 11 cases, where the OUSPG had an originating role, and 17 cases, where it was a tracking or contributing role. A checkpoint based summary of the database subset is presented in Appendix B.

Observations were gathered on different activities and stages in the vulnerability life-cycle. Problem areas pertaining to a large variety of cases are illustrated below.

3.1. Overall observations

Communication delays are inherent during all phases of the vulnerability life-cycle. Factors such as different time zones, holidays, the workload of the actors and the number of parties involved all contribute to the observed lag. Large delays may lead to misunderstandings, and thus confirmation should always be sought via alternative channels before rash conclusions or decisions are made.

One of the surprising perplexities involved in the vulnerability life-cycle is the accelerated time frame phenomenon. The originating party may be best prepared and oriented to support the life-cycle near the time of discovery, verification and reporting of the vulnerability. On the other hand, it may take time for the repairing party to allocate resources for the vulnerability case. As a result, a case considered dormant

for months by the originators may reactivate suddenly and demand patch evaluation within a very short time frame, for example.

Safe communication channels should not be taken for granted, and requiring them may complicate the reporting phase. Even if the report recipients are prepared for an encrypted email exchange, two different standards, i.e. PGP and S/MIME, may be involved. If the initial contact is an encrypted email that requires more than usual effort to handle, it may be dropped on the floor due to overload at the receiving end. The overhead for initiating communication should be kept at a minimum, but it may be wise to avoid any exchange of potentially harmful information prior to the establishment of a trusted channel.

The collecting of tracking information should commence from day zero. Everything may start as "just one vulnerability" but later on turn into a unmanageable mess. The originating party should assign a unique tracking number to each vulnerability and collect tracking information given by the coordinators and repairers. Advisories, patches, reports and other communications related to a case should be archived for later perusal.

Multi-vendor cases should be identified as early as possible. Vulnerabilities bearing similarities in their manner of activation or exploitation should be grouped under the same case. For example, an advisory may be issued about a web browser with a buffer overflow type vulnerability in displaying an image with a long name tag. In such a case competing browser vendors are left in an awkward position if their products exhibit the same vulnerable behaviour and are not considered in the advisory. Release of the vulnerability details tends to stimulate testing of similar products, and findings may be presented in vivid discussions on full disclosure mailing lists. A shared code base foreshadows a multi-vendor discovery, in that the same vulnerable code could have been inherited by several branches of operating systems, or the same algorithm library could have been used in several competing products. An isolated code base does not exclude multi-vendor characteristics, since certain operations implemented in code may be more vulnerability prone than others. The same vulnerability may be repeated in independent implementations of the same conceptual functionality. Vendor-independent coordinators are best equipped to ensure proper identification of the scope of a vulnerability.

Prioritisation disagreements may arise during impact assessment. For example, the issue of local vulnerabilities exploitable via existing local user accounts vs. remote vulnerabilities exploitable without an existing account is controversial one. In the case of multi-user time-shared operating systems such as the UNIX variants, arguments may favour the comparable seriousness of both cases. The vast possibilities for compromise of a local unprivileged account pose a serious risk when combined with a local vulnerability permitting privilege elevation.

The communication overhead with the repairing party is considerably reduced if the actor in the repairing role has a software vulnerability tiger team, such as a security group, that can either participate in repairing or assume a coordinating role inside the vendor’s organisation. Vendors should benefit from the accumulated knowledge of a specialised group inside their own organisation.

3.2. Observations on vulnerability research

Identifying vulnerabilities may be considered as reverse engineering activity. Any legal concerns arising from strict license agreements or outstanding non-disclosure agreements should be considered during vulnerability research, and recipients of the reports should be agreed upon by the holders of such rights. When in doubt, disassemblies and execution traces should not be collected and should be omitted from the report, even though these are valuable for pin-pointing the vulnerability location. A less invasive approach is offered by research focusing on observed vulnerable behaviour.

3.3. Reporting related observations

The efficiency of the life-cycle is built up on the vulnerability report. The report has two conflicting requirements: completeness and compactness. It should provide sufficient details to support other activities in the process. A structured vulnerability report template is presented in appendix A.

FIRST teams such as the AusCERT and CERT-CC may be able to act as coordinators [8] [9] [10]. Response teams are familiar with different operating environments and vulnerability history and have comprehensive vendor contacts. These attributes provide a solid foundation for impact assessment, identification of repairers and initiation of communication with them. The CERT-CC charter promotes contact with them in cases requiring vulnerability coordination [11].

When a report is received by the vendors it may be directly entered in bug, problem or escalation databases. An unfiltered database entry would contain sensitive information such as potential exploitation details. Such a disclosure may be more likely if front-line support is used as the reporting channel, or if an automatic bug report filing system exists. These databases may be public, i.e. distributed to customers as a service or offered as a resource for a joint development effort. Care should be taken that no information not intended for publication is entered in such databases. Database clean-up after a leakage may be difficult or impossible due to factors such as CDROM distribution of the database snapshots.

In an attempt to trim down the number of messages exchanged, the vulnerability report may include coordination details, remarks and background information relevant for the parties directly involved in the vulnerability process but irrelevant for the actual repair process. Information that should not be carved in stone in the databases of the repairers should be clearly marked as such, see Appendix A for an example convention.

A software package may be popular but still remain unmaintained for a long time, e.g. the popular email package Message Handler (MH). Locating the proper repairers for cases involving unmaintained packages may prove to be non-trivial, as the originators or coordinators may have to assume the repairing role and face the problem of integrating the patch into package distribution. In cases where a proper vendor with the capability for generating a patch and integrating it into future releases can be identified, the repairing role should be assumed by such an entity.

A demonstration of exploitability may be required before the repairers can commence patching. Customising the exploit details, such as the padding, for yet another buffer overflow type vulnerability may appear to be a waste of effort, but it may be necessary for satisfactory completion of the case life-cycle. Public disclosures where the details are left as "an exercise for the reader" may result in insufficient pressure for patch generation but a disclosure sufficient for potential abusers to complete the exploit. Practice appears to indicate that most potential vulnerabilities disclosed publicly will attract a corresponding exploit in the long run.

Standard support channels provided by vendors may prove to be tedious for vulnerability reports. Front-line support may require pay per call for customers without a service contract, and some vendors may only take very limited WWW-form bug reports. Some vendors have adopted a policy of not requiring a support contract for security cases. FIRST teams may prove to be invaluable for tackling this problem, due to their existing contacts.

The technique of proposing a specific fix in the report may backfire. The repairers may directly adopt the proposed fix in order to meet the customer's desires, incorporate it blindly and verify it against the specific exploit, while a slight variant of the exploit would still result in penetration.

3.4. Observations on acknowledgements

A lack of feedback from vulnerability reproduction failures can cause considerable grief. Feedback should be supplied if the reproduction fails. The originators should attempt to identify the cause of the failure and to reiterate the technical details. An understanding of reproduction failures may reveal valuable work-around information.

3.5. Repair failures

Patch production can be a complicated and time-consuming process for the repairers. Factors such as multiple platforms, revisions, regression testing, available resources, interface change requirements and cost-benefit issues are apt to give rise to delays, and may even halt the mending activity. The originator should be provided with reasonable feedback on progress.

The patch quality may not meet the expectations of the originating party. The patch may address the problem at a specific line of source code, but a new vulnerable condition may occur a bit further down. This highly localised approach to patching has been called the *+/- 4-lines syndrome*. Also, different teams may be responsible for maintaining current releases and for integrating the fix into future releases. The people with the least security experience or with insufficient resources for a more thorough study may well be occupying maintenance positions.

The patch generation process may be excessively customer driven. A customer has to demand the fix and has to accept it before it is published. This may lead to delays in patch generation and may leave lingering problems in the case of communication breakdowns.

3.6. Observations on patch release

The originators may seek to establish the initial communication with minimal overheads, and thus provide only some tentative vulnerability details, with a deeper analysis to come. If a patch is released without granting the originating party a chance to verify the patch, it may lead to further vulnerability reports that could have been handled in the same case and with less overheads.

Some vulnerabilities may prove to be too extensive to patch in existing releases, and a work-around has to suffice until the next release. Problematic change requests include ones where public interfaces have to be altered in a way that interferes with their compatibility.

3.7. Observations on advisory generation

An understanding of the underlying priorities, mechanics and procedures in the advisory process will help the originators and repairers to collect and provide relevant information. McMillan (1996) covers such aspects from the coordinator perspective [3].

Draft advisories should be clearly marked as such, in case the material leaks. At the draft stage the advisory may contain unconfirmed and incomplete details that may cause confusions and inconvenience if released prematurely. Such an event occurred recently in the IBM vs. ssh case [12].

3.8. Observations on advisory release

A patch release may be comparable to a public disclosure. Descriptive "readme" files accompanying the patch may reveal explicit vulnerability details. Patch access is sufficient for reverse engineering of the actual failings, whereas comparison of the vulnerable version with the patched version may reveal the details. When release of an advisory is delayed relative to the patch there may be confusion as to whether the patch really addresses an already known problem or not.

The proposed metric of choosing the greater value of patch delay and advisory delay for measuring the curing delay may appear to be a harsh requirement. However, a leak of information, and the uncertainty arising from the bare patch release justify the strict line. When the proper patches are available, there is a need for an accompanying advisory. This may consist of just a short statement about the security nature of the patches with reference to related public disclosures. The most concise form observed has been an entry on the recommended security patch list.

Without a corresponding advisory, the patch may not reach sufficient penetration amongst customers. Even security-conscious customers may be reluctant to change working systems if the security aspect is not stated clearly enough. Patch deployment may be problematic, and the release of vulnerability information should be weighted up as a risk, especially when widely deployed and mostly unadministrated client system software is in question. Patch deployment tools may be non-existent or lacking.

Minimalistic patches should be provided for customers who do not wish to upgrade a whole release or to install new functionality bundled with the patch. Cases where a security patch has added new functionality, later identified as a source of new vulnerabilities, are not unheard of.

4. Discussion

The complete tracking data available for cases originating from the OUSPG support the proposed model. The cases unambiguously passed the check-points and the actors involved in the process assumed the roles identified here. The model served as a communication vehicle, brought discipline to vulnerability case management and gave a basis for systematic vulnerability data collection.

Although the proposed model and related discussion may appear overly concerned with performance and formalism, this may be justified. A disciplined approach could lessen the high risks involved if the vulnerabilities endanger vital portions of the infrastructure. The process of counter-balancing the cost of addressing the discovered vulnerabilities against the potential loss may appear to be a complicated one and should be based on solid risk management. Both threats and vulnerabilities are required for a risk to materialise. Geer (1998) presents an excellent discussion of risk management in security research and cost-benefit-risk trade-off [13].

Public disclosures are constantly taking place and penetration testing is being conducted by uncoordinated parties. Tiger teams carrying out systematic testing and handling vulnerability cases professionally should be able to reduce considerably the risk exposure caused by public disclosures. Such teams would represent a shift from reactive to proactive vulnerability work.

The motivation for vulnerability tiger team work could stem from a combination of academic, customer and vendor perspectives. Examples of academic vulnerability work could encompass the accumulation of vulnerability knowledge and aim beyond the penetrate-and-patch paradigm. The customer approach could include acceptance testing of commercial off-the-shelf products and other 3rd party software. The vendor perspective could emphasise measures for preventing the introduction of vulnerabilities into software products.

Repairers may be faced with the dilemma of accumulating vulnerability knowledge. Originators may be experts in the vulnerability domain and have resources that concentrate on technical aspects, whereas the parties in the repairing role may be product engineers with minimal resources for familiarising themselves with up-to-date vulnerability knowledge. Such perils may be cast aside by vendors that have tiger teams of their own.

Legal concerns such as the WIPO treaty may in the long run hinder software penetration testing conducted by vendor-independent parties by restricting activities considered to constitute reverse engineering [14]. The limitations imposed by laws and license agreements should be considered in the context of vulnerability tiger team activity.

Impact assessment is a complicated aspect of the vulnerability life-cycle. Each actor

must perform his own impact assessment for internal go/no-go decisions, although some media-sensitive cases may call for public impact assessment, which can best be presented by a party independent of the originating and repairing roles.

As well as being sources of public impact assessment, independent coordinators such as the FIRST teams have an important role in the vulnerability process. The identification of multi-vendor cases, the building of test laboratories, the reduction of communication overheads and the improvement of process efficiency are important tasks that are best conducted by vendor-independent parties. If economically feasible, these teams are encouraged to undertake and continue the vulnerability work.

4.1. Future work

The concept of forming vendor-independent vulnerability tiger teams should be evaluated and a model should be developed for their operation. A call for such action on public collaboration was voiced by Eric Allman (pers. comm. 1999).

The lack of widely deployed tools to support software penetration testing should be addressed, and the possibility of using licensing terms for such tools to enforce a grace period before public disclosure should be discussed in order to ensure the involvement of the repairing party.

Vulnerability tracking information should be elaborated, together with a classification of the vulnerabilities. Several classification schemes and taxonomies have been proposed by the research community, and a comprehensive study of vulnerability taxonomies has been conducted (Krsul 1998) [4]. Combining the performance statistics with vulnerability attributes would provide a basis for a more profound analysis of the case properties. Attributes could include characteristics such as the amount of communication, in terms of the number of emails required, impact classification and the technical type of the vulnerability.

Reflections on the case study should be collected into a concise and easily available document, which should provide a few rules of thumb for actors entering the vulnerability scene for the first time.

5. Conclusions

This study was designed to support the vulnerability process by encouraging a disciplined approach and by highlighting experiences gained by a vulnerability tiger team. An undisciplined process may be hard to understand, control and improve. Regardless of its apparent bias, this study aims to provide insights into the vulnerability process and proposes a systematic approach. Support is provided for vendor security contacts, both current and new members of the FIRST, and other interested parties, such as the media. The material presented here is mainly for the benefit of those entering the scene as the originating parties of vulnerability reports, and it should also fuel further discussion about this particular area of security work.

A terminology and a life-cycle model with primitive metrics were also presented above, a case study carried out and observations on it recorded. The process was found to be complicated but manageable.

The applicability of these findings is limited by the focus on the role of vulnerability tiger teams. Complicated aspects of vendor and security organisation involvement in the vulnerability process were abstracted by means of a black box approach. A further constraint is placed on generalisation of the findings by the OUSPG-centric nature of this discussion.

The observations emphasise the role of FIRST teams as coordinators and argue for the promotion of a shift from undisciplined, reactive vulnerability work towards a professional, proactive approach.

Further research on elaborating the case attributes is recommended in order to support analysis of the properties and behaviour of different cases. Tools and documentation to support preventive vulnerability work are called for. Actors involved in future cases are encouraged to treat these fragile aspects with care.

6. Acknowledgements

The authors of this document wish to acknowledge Olivier Malhomme for collecting the life-cycle data and for the modelling performed during his stay at the OUSPG. Enormous gratitude is expressed to AusCERT, CERT-CC and the respective vendors for their active role in resolving the cases originated by the OUSPG and for their valuable feedback. The input from Rauli Kaksonen (VTT Electronics), Danny Smith (Sun Microsystems) and Rob McMillan (AusCERT) is acknowledged. Last but not least, ideas and experiences donated by the community at large, such as the contributors to the full disclosure mailing list discussions, have been invaluable in the preparation of this paper.

7. References

[1] OUSPG "Secure Programming Group, University of Oulu, Finland (OUSPG)" <http://www.ee.oulu.fi/research/ouspg/> (Accessed 3 Feb, 1999)

[2] Sanchez, Miguel J. "Opening The Vendor Black Box". Opening talk in the vendor panel. The 9th Annual FIRST Conference and Workshop on Computer Security Incident Handling and Response, Bristol, England. June 23-27, 1997.

[3] McMillan, Rob. "Vulnerability/Advisory process". Workshop session. The 8th FIRST Conference and Workshop on Computer Security Incident Handling and Response, Santa Clara, California, United States. July 28-31, 1996.

[4] Krsul, Ivan. Software Vulnerability Analysis. PhD Thesis. Purdue University, Coast TR 98-09. 1998.

[5] IEEE. ANSI/IEEE Standard Glossary of Software Engineering Terminology. IEEE Press. 1990.

[6] Netspace Project "BUGTRAQ@NETSPACE.ORG" <http://www.lsoft.com/scripts/wl.exe?SL1=BUGTRAQ&H=NETSPACE.ORG> (Accessed 23 Feb, 1999)

[7] Cooper, Russ, "NTBugtraq - NTBugtraq Home" <http://ntbugtraq.ntadvice.com/> (Accessed 23 Feb, 1999)

[8] FIRST "Forum of Incident Response and Security Teams" <http://www.first.org> (Accessed 13 Mar, 1999)

[9] AusCERT "Australian Computer Emergency Response Team" <http://www.auscert.org.au> (Accessed 13 Mar, 1999)

[10] CERT-CC "Cert Coordination Center" <http://www.cert.org> (Accessed 13 Mar, 1999)

[11] Cert-CC "About the CERT(r) Coordination Center" <http://www.cert.org/meet_cert/meetcertcc.html> (Accessed 23 Feb, 1999)

[12] SSH Communications Security Ltd. "SSH - Products - SSH Protocols - Rootshell attack" <http://www.ssh.fi/sshprotocols2/rootshell.html> (Accessed 3 Feb, 1999)

[13] Geer, Daniel E. Driving the future. ;login: Security Special Issue. May 1998. pp.24-33.

[14] Electronic Frontier Foundation "Intellectual Property - WIPO Copyright & Database Protection Proposals Archive" <http://www.eff.org/pub/Intellectual_property/WIPO/> (Accessed 3 Feb, 1999)

APPENDIX A: A Vulnerability Reporting Form

This appendix describes the vulnerability reporting form adopted by the secure programming group at the University of Oulu (OUSPG). The structure of the form is illustrated section by section, with accompanying comments.

The following shorthand notation is adopted:

- <ACTOR-ID> is used to denote the actor responsible for triggering the event. Typical actors used by the OUSPG include: OUSPG, AusCERT, CERT-CC and respective vendors.
- <CONTACT> is used to denote the contact details for the actors involved in the vulnerability case. This is typically the email address of the party involved.
- <COUNT> is used to denote a quantity, e.g. the number of emails exchanged during the process of resolving a vulnerability case.
- <DATE> is used to denote the timestamp of an event, a checkpoint in the vulnerability process, in the format YYYYMMDD. For example, 19990124 represents 24th January 1999.
- <DESC> is used to denote a short free-form description of the entry or the title of a related document.
- <STATE> can be either PENDING, OPEN or CLOSED. Ideally this should denote the status of the vulnerability case by identifying the current stage in the process.
- <TRACKING#> is used to denote the case identifier for tracking purposes. OUSPG, AusCERT and CERT-CC, for example, use their own tracking numbers, and respective vendors may have one or more tracking identifiers in the form of service call numbers, escalation numbers or bug identifiers.
- The symbol [*] at the end of a line is used to denote repetition (N = 1). Fields marked with this symbol may appear once or several times in order to supply all relevant information, e.g. when several bug identifiers have been assigned by different vendors.

Completeness of the information entered in certain sections is promoted by check-box type input lists covering most typical entries. For brevity, some check-box lists have been collapsed into sample entries. Some formatting aimed at improving readability has been removed in order to save space.

A.1: Title information

The title section introduces the critical information at the top of the document. A unique vulnerability case identifier and list of recipients for the vulnerability report are included.

```
             ----------------------------------------
             University of Oulu Secure Programming Group
                 Vulnerability Reporting Form (V1.3)
             ----------------------------------------
  OUSPG#:               <TRACKING#>  <DESC>
  Distribution:         <CONTACT> [*]
```

A.2: Preface

The preface contains information that should not be entered in persistent databases such as those kept by vendors. This may include sensitive material about 3rd party vulnerabilities or merely personal remarks. All such information is collected here and clearly marked in order to facilitate safe and easy data entry by recipients.

```
0. Preface [please omit from persistent databases]
  <DESC> [*]
```

A.3: Tracking information

Initial tracking information is included in the report as background material and as a reminder of the vulnerability process check-point structure. The tracking information should be updated later in a separate database. Most date-based events may contain several entries and for multiple events the earliest appropriate date is used in the statistics.

```
1. OUSPG tracking Information
  OUSPG#:               <TRACKING#>
  CERT#:                <TRACKING#> [<ACTOR-ID>] [*]
  BugID#:               <TRACKING#> [<ACTOR-ID>] [*]

  Advisory:             <DESC> [<ACTOR-ID>] [*]

  Discovered:           <DATE> [<ACTOR-ID>] [*]
  Verified:             <DATE> [<ACTOR-ID>] [*]
  Reported:             <DATE> [<ACTOR-ID>] [*]
  Acknowledged:         <DATE> [<ACTOR-ID>] [*]
  Fix available:        <DATE> [<ACTOR-ID>] [*]
  Public disclosure:    <DATE> [<ACTOR-ID>] [*]
  Public exploit:       <DATE> [<ACTOR-ID>] [*]
  Advisory date:        <DATE> [<ACTOR-ID>] [*]

  Type:                 <DESC> [*]
  Category:             <DESC> [*]

  Emails:               <COUNT> [<DATE> <ACTOR-ID>] [*]
  State:                <STATE>
```

A.4: Affected product and vendor

The affected product section aims to provide sufficient details to enable front-line service or process coordinators to select the right vendor, product engineering or escalation team to address the problem. The distribution and installation information represents an attempt to provide a starting point for impact assessment conducted by the actors in a coordinating role.

```
2. Affected Product
   2.1. Product or Package Name: <DESC>
   2.2. Short description of the affected product: <DESC>
   2.2. Vendor(s): <ACTOR-ID> [*]
   2.3.        Distribution:
       [ ] Bundled with the core OS
       [ ] Purchased or acquired separately
   2.4. Installation:
       [ ] Installed by default in a typical environment
       [ ] Installed by default but has to be activated by admin
       [ ] Installed and activated separately by admin
       [ ] Installed and activated separately by an unprivileged user
   2.5. Product available for OS versions: <DESC> [*]
```

A.5: Suspected Impact and Vulnerability Type

The section on the suspected impact and vulnerability type gives further details to support impact assessment and attempts to classify the vulnerability type.

```
3. Suspected Impact and Vulnerability Type
   3.1. Exploitability:
       [ ] Local - by users with local accounts
       [ ] Remote - over the network without an existing local
account
   3.2. Compromise:
       [ ] Total compromise of a privileged account    UID: <DESC>
       [ ] Total compromise of an unprivileged account UID: <DESC>
       If not a total compromise, please specify:
       [ ] Confidentiality
       [ ] Integrity
       [ ] Availability (Denial of Service)
   3.3. Timing:
       [ ] Exploitable at any time
       [ ] Exploitable under specific conditions: <DESC>
   3.4. Vulnerability Category:
       [ ] Buffer overflow
       [ ] File manipulation
          [ ] Ability to create user-modifiable arbitrary files
          [ ] Ability to create unmodifiable arbitrary files
          [ ] Ability to truncate or remove arbitrary files
          [ ] Ability to change owner of arbitrary dirs or files
          [ ] Ability to change protection modes of arbitrary dirs or
files
          [ ] Other: <DESC>
       [ ] Execution
       [ ] Exploitation involves a time window of the race condition
type
       [ ] Other: <DESC>
   3.5. Comment on impact: <DESC>
```

A.6: Tested Software

The tested software section is crucial in order to support reproduction of the vulnerability by the coordinators and repairers. The environment, including platform and operating system versions, should be described in sufficient detail, and the base version and installed patches of the subject software used for verifying the vulnerability should be described in detail. Since the same code may be shared

between different versions of the same or related products, already tested variants should be identified. Since the same vulnerability may re-emerge or other complex interrelationships with different vulnerabilities may exist, related bugs should be identified.

```
4. Tested Software
  4.1. Test Environment: <DESC>
  4.2. Software Package Version: <DESC>
  4.3. Related patches
    [ ] No related patches were available to the author
    [ ] Patches for the product used during testing: <DESC>
  4.4. Other versions
    [ ] No other versions were tested
    [ ] Other versions found vulnerable: <DESC>
    [ ] Other versions found immune: <DESC>
  4.5. Possibly related known bugs:  <DESC>
```

A.7: Vulnerable Code

The vulnerable code section is repeated for each of the vulnerabilities involved in the case. Precise identification of the subjects, the method by which the vulnerability was discovered and a description of the anomalies found and their location are desirable.

```
V1. Vulnerable Code #01 [*]
  V1.1. Binary identified to exhibit vulnerable behaviour: <DESC>
  V1.2. Binary Type
     [ ] setuid / setgid
     [ ] network (privileged) daemon
     [ ] local (privileged) daemon
     [ ] other
  V1.3. Functionality Description (’man’ abstract): <DESC>
  V1.A1. Anomaly #01 [*]
    V1.A1.1. Anomaly discovered in:
      [ ] Normal system operation
      [ ] Casual inspection
      [ ] Stress-test to find vulnerabilities
      [ ] Other: <DESC>
    V1.A1.2. Anomaly discovered via:
      [ ] Reported by a 3rd party:
      [ ] Suspicious filenames
      [ ] Unexpected crash
      [ ] ’sotruss’, ’truss’ or ’par’ output
      [ ] Debugger
      [ ] Custom Vulnerability Tool: <DESC>
      [ ] Other: <DESC>
    V1.A1.3. Anomaly description
      While running: <DESC>
      Tool output: <DESC>
      Explanation: <DESC>
   V1.A1.4. Anomaly location
      [ ] Unknown
      Stack-trace (’pstack’ - if available): <DESC>
      Process-map (’pmap’ - if available): <DESC>
      [ ] Vulnerable code is located inside the core executable
      [ ] Vulnerable code is located in a library: <DESC>
   V1.A1.5. Comment: <DESC>
```

```
  V1.5. Comment: <DESC>
```

A.8: Exploitation details

A proof of concept should be given for each anomaly by providing sufficient exploitation details.

```
E1. Exploitation details - for V1.A1 [*]
  <DESC>
```

A.9: Work-around

If any work-arounds have been found by the originators, these should be identified and their side-effects and efficiency assessed.

```
W1. Work-around [*]
  W1.1. Proposed work-around
    W1.1.1. Description
    [ ] Removal of setuid/setgid bit
    [ ] Other: <DESC>
    W1.1.2. Side-effects
    [ ] Unknown
    [ ] Other: <DESC>
    W1.1.3. Efficiency
    [ ] Unknown
    [ ] Prevents the presented exploit
    [ ] Other: <DESC>
    W1.1.4. Detailed description: <DESC>
```

A.10: Supplementary information

Supplementary information such as overall comments, sufficient and redundant contact information and possible appendices or attachments are given at the end of the report.

```
5. Comments: <DESC>
6. Contact information: <DESC>
7. Attachments: <DESC>
```

APPENDIX B: Summary of the Vulnerability Case Study

A summary of a subset of the vulnerability case database collected by the OUSPG is presented in Table B.1. Selection criteria for the cases presented were the availability of both a patch and an advisory. Dates given for different checkpoints are given in accordance with the proposed model. Furthermore, a field with the number of emails exchanged by AusCERT in a coordinating role has been added to illustrate enhanced attribute collection. Since the cases are not comparable, all vendor references have been abstracted in order to avoid "vendor bashing".

Table B.1 A summary of a subset of the OUSPG Vulnerability Database

OUSPG#	DISCOVERY DATE	VERIFICATION DATE	REPORTING DATE	ACK. DATE	PATCH DATE	ADVISORY DATE	EMAILS
0001	-	19960902	-	-	19960710	19960724	?
0002	-	19960816	-	-	19961010	19960730	40
0003	-	19960816	-	-	19960916	19960726	?
0004	19960916	19960919	19960919	19960923	19970110	19970501	33
0005	-	-	-	-	19961223	19960802	80
0006	19960807	19960808	19960808	19960809	19961010	19961015	80
0007	19961010	19961014	19961015	19961017	19961029	19961120	67
0008	19960817	19960819	19960819	19960819	19961223	19970224	33
0009	-	-	-	-	19970401	19970324	?
0010	-	-	-	-	19970409	19970226	?
0011	-	-	-	-	19970511	19970213	28
0012	19970423	19970423	19970423	19970424	19970510	19970528	17
0013	19970301	19970301	19970304	19970305	19970421	19970604	31
0014	-	-	-	-	19970530	19970604	?
0015	19960528	19960528	19960601	19960808	19961010	19970625	128
0016	19961006	19961006	19961007	19961007	19970617	19970624	7
0017	-	-	-	-	19970612	19970624	?
0018	-	-	-	-	19970702	19970715	?
0019	-	-	-	-	19970711	19970117	?
0020	-	-	-	-	19970718	19970730	?
0021	-	19970520	-	-	19970707	19970521	33
0022	-	-	-	-	19970811	19970812	?
0023	-	-	-	-	19970716	19970825	?
0024	-	-	-	-	19970428	19970820	?
0025	19960903	19960905	19960905	19960906	19970813	19970917	7
0026	19970520	19970523	19970523	19970523	19971006	19971028	4
0027	-	-	-	-	19970826	19971203	17
0028	19960818	19960818	19960819	19960819	19970113	19980121	20